

Java-Magazin 10/2001

Daten frei Haus -Ein Servlet als Alternative zu JSP

von Andreas Tengicki und Henry Wolf

Für die Generierung von dynamischen HTML-Seiten gibt es verschiedene technologische Ansätze. Im folgenden wird ein Projekt beschrieben bei dem die Realisierung mit Servlets erfolgte.

In diesem Artikel soll erst konzeptionell und dann an Hand eines realisierten Projekts die Möglichkeiten der dynamisch-datenbankgestützten Generierung von HTML Seiten mit einem Servlet gezeigt werden. Neben der Möglichkeit mit Servlets zu arbeiten, gibt es konzeptionelle Alternativen, an erster Stelle natürlich Java Server Pages (JSP) und Perl.

Bei JSP werden HTML-Dateien verwendet, die in einem speziellen Tag Java Quellcode enthalten sein können. Ein entsprechender WebServer, bzw. ein Servlet eines WebServers, übersetzt das Java der Seiten beim ersten Aufruf und liefert dann das HTML, welches sich aus dem HTML Anteil und dem in Java gehaltenen Teil der Vorlage ergibt. Der Vorteil dieser Variante ist, daß sich dynamische Inhalte relativ einfach in eine Webseite integrieren lassen, Web-Designer und Java-Entwickler können sich jeweils auf ihre Aufgabengebiete konzentrieren.

Davon ausgehend, daß bei der Generierung dynamischer HTML-Seiten wesentliche Inhalte aus einer Datenbank kommen, haben wir einen Alternativen Ansatz verfolgt, wir nennen es SQL Server Pages (SSP). Dabei können SQL Statements direkt in die HTML Seite integriert werden und mittels HTML formatiert ausgegeben werden. Eine Integration der Daten in das System ist nicht erforderlich. Es werden die Daten direkt von der vorhandenen Datenquelle entgegengenommen. Die einzelnen Elemente dieses Systems sollen im folgenden beschrieben werden.

1. Aufbau des SSP Servlets

Der WebServer leitet die Anfrage an das System weiter, zur Realisierung wurde wegen der Plattformunabhängigkeit ein Java Servlet gewählt. Dabei wird die http-Anfrage an das Servlet übergeben, welches dann einen HTML-Ausgabe-Stream erzeugt. Die schematische Abfrage zeigt Listing 1.

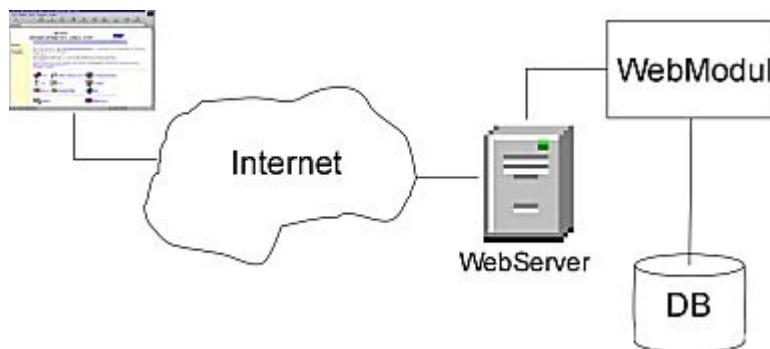


Abb.: 1 – Behandlung einer http-Anfrage mit Servlet

Listing 1: Behandlung der Anfrage im Servlet (Auszug)

```
package sdctec.webengine;

import ... /**
 *Main Servlett Class
 *generates HTML-Pages based on a database
 *
 *@author SDCTec GmbH-tg
 */

public class webengine extends HttpServlet
{
    private static Hashtable configurations = new Hashtable();
    private String cfgFileName = null;
```

```

private JDBCConnection dbCon = null;
private CfgFileBase cfgFile = null; //Objekt für den Zugriff auf die Daten

public void init(ServletConfig config)
    throws ServletException
    {
        super.init(config);
        cfgFile = new CfgFile(getInitParameter("configFile"));
    }

/**
 *answer for any Servlet hppt-get request
 *@param req current req
 *@param reso response for the answer
 */

private void handleRequest(String path, boolean content,
    HttpServletRequest req, HttpServletResponse resp)
    throws Exception
    {
        initDB(req,resp);
        //Bestimmen einer Session, wenn noch keine Id dann ist
        //dies neu,
        SessionEnvironment session =
            new SessionEnvironment
                (cfgFile,req.getSession(false),req,resp);
        //analyse des QueryStrings
        if (req.getQueryString() != null)
            session.putList(HttpUtils.parseQueryString
                (req.QueryString()));

        //Analyse der Form-Paramter
        if(content)

            session.putList(HttpUtils.parsePostData
                (req.getContentLength().req.getInputStream()));
        //Vorbereitungen für die Ausgabe treffen
        resp.setContentType("text/html");
        PrintWriter out = new PrintWriter
            (session.getResponse().getOutputStream());
        //Jetzt den loStream zur Anfrage bestimmen
        InputStream ins = dbData.getStreamViaPid
            (session.getString("pid"));

        //Ausgabe des Ergebnisses
        out.print("<!DOCTYPE HTML PUBLIC \"-
            //W3C//DTD HTML 4.0//EN\">\n");
        out.print("<!--webengine \"SDCTEC WebEngine-
            "+cfgFile.getProperty("servlet.id")+"\"-->\n");
        XMLVorlagenHandler xml = new XMLVorlagenHandler
            (out, session);

        xml.startHandler();
        xml.TransferParser(new TemporaryPrinter(ins).getReader());
        xml.stopHandler();
        out.flush();
    }

public void doGet (HttpServletRequest req,
    HttpServletResponse resp)
    throws ServletException

```

```

{
try
{
    handleRequest("getpage",false,req,resp);
}
catch (Exception e)
{
    Logbuch.errorLog(e,null);
    throw new ServletException(e.getMessage());
}
}

public void doPost(HttpServletRequest req,
                    HttpServletResponse resp)
throws ServletException
{
try
{
    handleRequest("sendpage",true,req,resp);
}
catch (Exception e)
{
    Logbuch.errorLog(e,null);
    throw new ServletException(e.getMessage());
}
}
}
}

```

2. Aufbau des SSP Servlets

Servlets unterstützen einem zudem im Session Management, d.h., die eigentlich zusammenhanglose Abfolge von HTTP-Anfragen kann in einen Kontext gestellt werden. Dies ist zum Beispiel für die Realisierung eines Shops unabdingbar, denn es muß sichergestellt werden, daß ein Warenkorb auch einem Benutzer zugeordnet werden kann. HTTP als Internetprotokoll ist zustandslos, eine Anfrage steht in keinem Zusammenhang zu einem vorigen. Zwei Wege, dies zu umgehen, sind möglich. Ein häufig verwendeter sind Cookies, dabei werden (kleine) Variablenwerte im Browser (Client) des Anfragenden hinterlegt und bei jeder HTTP-Anfrage an den Server übermittelt, so daß dieser den Zusammenhang aufbauen kann. Eine Alternative dazu stellt URL-Rewriting dar. Dabei werden alle Links auf weitere Seiten vom Server mit einer SessionID in der URL erweitert. Dies setzt allerdings voraus, das alle Links einer Webanwendung dynamisch erzeugt werden, damit die Ergänzung der SessionID vorgenommen wird. Die Servlet API (JSDK 2.1) stellt eine Funktion zur Verfügung, die diese Ersetzung für einen vornimmt. Im Projekt ist dies dann wie folgt realisiert.

```

url = <url ohne SessionID>
url = session.getResponse ().encodeURL (url);

```

3. Hierarchisches Seitenkonzept

Zur Generierung des eigentlichen HTML-Textes muß das Servlet nun die Vorlage bestimmen, lesen und interpretieren. Das Ergebnis der Interpretation ist dann die HTML-Seite für den Browser des Benutzers der Webanwendung. Das Ergebnis der Interpretation ist dann die HTML-Seite für den Browser des Benutzers der Webanwendung. Bei der Speicherung der SSP-Seiten im System, haben diese einen hierarchischen Namen, z.B. *sdctec_dokus_javamag_art0101*. An Hand dieses Names wird nun ein Layout, also eine Vorlage für den Seitenrahmen und eine Vorlage für den Inhalt (Content) gesucht. Beispielsweise findet sich dann ein Layout mit dem Namen *sdctec* und ein Content mit dem Namens *sdctec_artikel_javamag_art0101*. Zuerst wird die Layoutvorlage interpretiert und mit dieser üblicherweise ein einheitlicher Seitenkopf und Seitenfuß, sowie ein Navigationsmenu erzeugt. Damit ist dann eine Frame-freie HTML Erzeugung möglich. Diese Vorlage enthält dann ein SSP-Tag `<content/>`, an dieser Stelle wird der eigentliche Inhalt der Webseite angezeigt. Zur Veröffentlichung eines weiteren Artikels ist es dann also nur notwendig, den neuen Content zu erfassen und mit sinnvoller Adresse in das System zu integrieren. Die Anzeige von einheitlichen Seitenkopf und -fuß etc., wird gewährleistet. Hieraus ergeben sich zwei Vorteile gegenüber JSP und vergleichbaren Technologien. Derjenige, der einen neuen Artikel veröffentlicht, braucht sich weder um die


```

public abstract void setEntityResolver (EntityResolver resolver);
public abstract void setDTDHandler (DTHandler handler);
public abstract void setDocumentHandler (DocumentHandler handler);
public abstract void setErrorHandler (ErrorHandler handler);
public abstract void parse (InputStream source)
    throws SAXException, IOException;
public abstract void parse (String systemId)
    throws SAXException, IOException;
}

package org.xml.sax;
 * @author David Megginson (ak117@freente.caleton.ca)
 */
public interface DocumentHandler {
    public abstract void setDocumentLocator (Locator locator);
    public abstract void startDocument ()
        throws SAXException;
    public abstract void endDocument ()
        throws SAXException;
    public abstract void startElement (String name, AttributeList atts)
        throws SAXException;
    public abstract void endElement (String name)
        throws SAXException;
    public abstract void characters (char ch[], int start, int length)
        throws SAXException;
    public abstract void ignorableWhitespace (char ch[], int start, int length)
        throws SAXException;
    public abstract void processingInstruction (String target, String data)
        throws SAXException;
}

```

5. Einbindung von Datenbanken

Hauptzielsetzung des Servlets war die Anbindung von Datenbanken an die HTML-Vorlagen. Dazu stellt das Servlet eine Datenbankverbindung zur Verfügung, auf die dann in den Vorlagen zugegriffen wird. Das bedeutet insbesondere, daß der Entwickler der HTML-Vorlagen sich nicht mehr um Details der JDBC-Verbindung kümmern muß. Durch den Einsatz von JDBC-Wrapper Klassen (vgl. JavaMagazin 11/2000) kann man erreichen, daß das eingesetzte SQL kompatibel zu InterBase, Oracle und ggf. noch weiteren Datenbanken wird. Die Datenbankverbindung wird in der Initialisierung des Servlets angelegt und aufrechterhalten. Die Details der Verbindung, wie zu verwendener JDBC-Treiber und Datenbankpfad werden in der Konfigurationsdatei des Servlets hinterlegt. Achten Sie bei der Verwendung von JDBC-Treibers auf die Details, auch wenn die Treiber jeweils die entsprechenden Interfaces implementieren werden, z.B. die Ergebnisse einer Stored Procedure bei InterBase über die Parameter und bei Oracle um ein Result Set zurückzugeben. Auch ist die Schnittstelle auf die Metadaten einer Abfrage, die ggf. die Typermittlung eines Rückgabewertes ermöglichen, höchst unterschiedlich implementiert. In der HTML Vorlage hat dann ein SQL Statement zur Bestimmung einer Adresstabelle den folgenden Aufbau, den Sie in Listing 3 sehen.

Listing 3: SSP-Vorlage mit SQL-Statement

```

<wbeDocument>
<content/>
<url>sdctec_javamag_example1</url>
<data>

<h3>Adressen</h3>
<sql style="read">
  <cmd>select name, strasse, ort, telefon from adressen order by name</cmd>
  <header>
    <table>
      <tr><th>Name</th><th>Strasse</th><th>Ort</th><th>Telefon</th></tr>

```

```

</header>
<text>
  <tr><td>$name</th><th>$strasse</th><th>$ort</th><th>$telefon</th></tr>
</text>
<footer>
  </table>
</footer>
</sql>
</data>
</wbeDocument>

```

Zur Behandlung dieses SQL-Statement im Servlet wird auf die entsprechenden Tags in der Vorlage reagiert. Dabei kann auf die vom Servlet zu Verfügung gestellte Datenbankverbindung (dbCon) zurückgegriffen werden. Es wird im wesentlichen immer auf das Ende-Tag reagieren (</cmd>) und die dann in *getSource()* zur Verfügung stehenden Daten des Tags genutzt. Die Procedure ist Teil eines eigenen DocumentHandlerObjects, welches auf Teil des für den Parser implementierten *DocumentHandler*-Objects, welches Teil des für den Parser implementierten Document Handlers ist und auf definierte Tags reagiert. Dieser Parser kann bei entsprechender Nachfrage im Java Magazin auch mal detaillierter beschrieben werden.

Listing 4: Behandlung eines SQL-Tags

```

public void parseEndElement (String tag, TemporaryPrinter source)
  throws Exception
{
  try
  {
    if (tag.compareTo("cmd")==0)
    {
      String cmd = source.getString();
      dbRes = new JDBCSQLException(dbData.getDbCon(),cmd,null);
      for (int i=0; i<stmt.getParamName(i);
      {
        String p = stmt.getParamName(i);
        String s = session.getString(p);
        if (s != null) stmt.setParam(i.s);
      }
      stmt.execute();
    }
    else if (tag.compareTo("header")==0)
    {
      if (!dbRes.getEOF())
        printSource(source);
    }
    else if (tag.compareTo("text")==0)
    {
      while (!dbRes.getEOF())
      {
        printSource(source);
        dbRes.next();
      }
    }
    else if (tag.compareTo("footer")==0)
    {
      printSource(source);
    }
    else if (tag.compareTo("sql")==0)
    {
      stmt.close();
    }
  }
}

```

```

//Hier erfolgt das erneute Parsenm des mit SQL erzeugten SSP-Codes
TransferParse(source);
}
}
catch (Exception e)
{
throw new Exception(e.getMessage());
}
}
}

```

Neben SQL-Statements zum Lesen von Daten (select) können natürlich auch SQL-Befehle ausgeführt werden (*insert*, *update*, etc.) und so Daten manipuliert werden. Zusammen mit weiteren Sprachelementen, die im Parser implementiert werden, wie zum Beispiel Bedingungen und Variablenverwaltung, ist die Implementation von ganzer Anwendung (Shop, Helpdesk etc.) möglich und auch bereits realisiert worden.

Dabei ist immer zu beachten, daß der Entwickler der Webanwendung ohne eigentliche Java Kenntnisse auskommt, sondern "nur" HTML und die Struktur seiner Datenbank die er ins Inter/Intranet stellen will.

6. FTP-Server zum Daten Im- und Export

Da das Servlet bereits einen vollständigen XML-Parser enthält, ist dann auch relativ einfach, eine Schnittstelle für den Daten Im- und Export anzubieten. Die SSP-Vorlagen werden konsequenterweise nicht in einem unübersichtlichen Dateibaum abgelegt, sondern in den speziellen Tabellen der Datenbank verwaltet. Wie aber kommen die Vorlagen da hinein?

Üblichweise werden HTML-Seiten auf einen WebServer mittels des FTP-Protokolls übertragen. Dabei werden zwei Socket Verbindungen zum WebServer aufgebaut, über eine Verbindung laufen die Befehle des FTP-Protokolls in ASCII und über die anderen die Daten in ASCII oder Binär. Clients zum Empfangen und Senden von Dateien via. FTP gibt es wie Sand am Meer und sind in die meisten Werkzeuge zum Erstellen von HTML-Seiten eingebaut. Wenn das Servlet nun einen FTP-Server enthält, der die Daten der Datenbank an so einen FTP-Client senden, bzw. von dort empfangen kann, ist die Aufgabe elegant gelöst.

Das Servlet ansich hat dabei nur eine kleine Aufgabe am Rande. Servlets beantworten http-Anfragen und haben mit dem FTP-Protokoll nichts gemeinsam. Man kann aber eine spezielle HTTP-Anfrage nutzen, um einen eigenen Thread zu starten, der einen SocketServer enthält um die FTP-Anfragen eines Clients zu beantworten. Das Servlet selbst enthält für die Beantwortung der HTTP-Anfrage keinen eigenen Socket, da es mit dem WebServer über Streams kommuniziert und die Behandlung der TCP/IP Sockets dem WebServer obliegt.

Wenn der SocketServer eine Anfrage eines entsprechenden Clients akzeptiert, wird ein FTPServer-Thread gestartet, folgender Auszug aus der Quelle soll die Funktionsweise des FTP-Servers veranschaulichen. Dabei ist zentraler Bestandteil, die Funktion *run*, die wie bei jedem Thread die eigentliche zu durchzuführenden Operationen enthält und verlassen wird, wenn die Socketverbindung beendet wird, oder zusammenbricht.

Listing 5: Teile eines einfachen FTP-Servers

```

public class FtpServerThread
extends Thread
implements ServerThread
{
...
public void run ()
{
try
{
out=new PrintWriter(con.getOutputStream (), true);
in=new BufferedReader(new InputStreamReader(con.getInputStream()));
inetClient = con.getInetAddress();
ClientIP = inetClient.getHostAddress();
out.println("220 SDCTecWebEngine FtpServer ready.");
int errorCount=0;
while(!Finished && errorCount < 3 && !isInterrupted())

```

```

{
//String str=in.readLine();
//readLine blockiert den Thread, so daß kein interupt durchkommt, aber
String str="";
char c;
do
{
while (!in.ready() && !isInterrupted())
sleep(50);
c=(char)in.read();
if (c != '\n')
str = str + c;
}
while (c != '\n' && !isInterrupted());
if (str==null&&!isInterrupted())
{
out.println("500 Command is null");
errorCount++;
}
else if (!isInterrupted())
try
{
//Abarbeitung der Befehle des FTP Protokolls gemäß RFC
if (str.startsWith("RETR")) cmdRETR(str.substring(4).trim());
else if(str.startsWith("STOR")) cmdSTOR(str.substring(4).trim());
else if(str.startsWith("TYPE")) cmdTYPE(str.substring(4).trim());
else ...
else if(str.startsWith("QUIT"))
{
out.println("GOOD BYE");
Finished = true;
}
}
else
out-println("502 Command not implemented");
}
catch (Exception e)
{
errorCount++;
}
}
closeAll ();
}
catch (Exception e)
{
Logbuch.errorLog(e, "FtpServerThread.end");
}
}

private void cmdRETR(String cmd)
{
try
{
out.priIntln("150 Binary data connection");
Socket t=new Socket(CLientIP,ClientPort);
OutputStream out2=t.getOutputStream();
ftpGetFile(out2,dir,cmd);
out2.close();
out.println("226 transfer complete");
}
}

```

```
t.close();
}
catch(Exception e)
}
Logbuch.errorLog(e,"cmdRETR");
}
}
```

7. Integration in Apache/Jserv

Wenn Apache mit Jserv installiert ist, läßt sich das Servlet, wie jedes andere in *zone.properties* einbinden. Einzig zu beachten, ist dabei der Zugriff auf die Datenbank. Da Apache/Jserv zur Sicherung des Servers, auf dem es läuft keinen, bzw. nur eingeschränkten Zugriff auf die Ressourcen des Systems haben, implementiert man den Zugriff auf die Datenbank am einfachsten über TCP/IP. D.h., die Adresse der Datenbank wird angegeben, als würde sie sich auf einem anderen Computer befinden, bei InterBase also z.B. *jdbc:interbase://freetown/c:/jsdk2.1/ibweb.gdb*. Befindet sich die Datenbank auf dem gleichen Rechner, wie der WebServer, verwendet man statt dessen halt *localhost*. Dabei wird dann auch die Trennung der einzelnen Komponenten des Systems deutlich.

Der Client des Benutzers kommuniziert via HTTP mit dem WebServer (Apache). Apache arbeitet via apj mit Jserv zusammen und nutzt dafür auch eine lokale TCP/IP Verbindung. Jserv wiederum greift gemäß den Spezifikationen des JSDKs auf das Servlet zu. Das Servlet zu guter Letzt kommuniziert dann über TCP/IP, auch ggf. lokal mit der Datenbank, im Falle von InterBase, geschieht das auf dem gds-Port 3050.

8. Fazit

Das beschriebene Servlet realisiert eine Technologie vergleichbar mit JSP. Es werden Vorlagen (SSP-Seiten) aus einem Verzeichnis, hier datenbankbaiserend, entnommen und interpretiert. Das Ergebnis der Interpretation sind reine HTML-Seiten für den Anwender der WebAnwendung in seinem Browser. Durch den Einsatz von Servlets, an Stelle von JSP konnten Anforderungen, die auf die direkte Anbindung von Datenbanken abzielen direkter unterstützt werden, als dies in ganz allgemeinen Lösungen wie JSP möglich ist. Zudem wird eine Trennung des Know Hows möglich und zwar in Java-Know-How Design & Datenbank-Know-How.