Vorlesungsreihe
Entwicklung webbasierter Anwendungen

Webanwendungen mit Java

Prof. Dr.-Ing. Thomas Wiedemann
email: wiedem@informatik.htw-dresden.de

HOCHSCHULE FÜR TECHNIK UND WIRTSCHAFT DRESDEN (FH)
Fachbereich Informatik/Mathematik

Gliederung

- Allgemeine Optionen unter Java
  - Überblick Client-Server-Anwendungen
- Webanwendungen mit Java
  - JavaServlets
  - Java Server Pages
  - Programmierbeispiele zu Servlets und Java Server Pages
  - Die Tomcat-Engine

Optionen der Datenverwaltung bei Webanwendungen

- in proprietären Formaten mit eigenen Dateien
- in Datenbanken mit relationalen oder objektorientierten Schema

Vorteile von Datenbanken :

- im Vergleich zu Dateiablagen mehrfach leistungsfähiger bzgl. Selektion, Einfügen und Manipulieren der Daten
- standardisierte Schnittstellen (ODBC, Datensprache SQL)
- fast alle betriebswirtschaftlichen Anwendungen laufen auf Datenbanken (direkte Einspeisung möglich und erstrebenswert ! -> der Kunde wird zum **Prosument** - - d.h. er produziert seinen Konsum-Verwaltungsprozeß selbst)

Nachteile

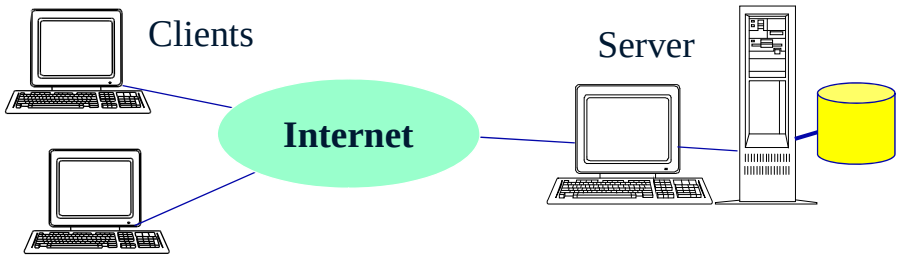
- wesentlich aufwendiger (Computerressourcen, notwendige Performance muß ca. 10-fach größer sein bei gleichem Anforderungsvolumen)
- echte Highspeed-Anwendungen laufen auch heute noch auf Dateibasis oder eigenen, speziellen Datenbankformaten

Fazit:

- Datenbankeinsatz sollte heute bei Neuentwicklungen als Standard definiert werden
- sinnvoll sind auch Mischformen: bei echter dynamischer Transaktion Ausführung von Datenbankaktionen mit nachfolgender Aktualisierung statischer Seiten (sinnvoll bei Anzahl_Lesezugriffe >> Anzahl_ Schreibzugriffe (vgl. Ebay !!))

Allgemeine Konfiguration von Webanwendungen

- Datenbank befindet sich generell auf dem Server, räumlich entfernt von den Clients



Zwei Grundkonzepte bezüglich der Konfiguration

1. Konzentration der gesamten Anwendungslogik auf Server, Clientrechner realisiert nur sehr einfache Anzeige und Visualisierungsfunktionen **(Thin-Client-Konzept) - unter Java mit Java Servlets oder Java Server Pages**
2. Anwendungslogik wird auf dem Client realisiert **(Fat Client)**, Server stellt nur Datenbankzugriff bereit -> **unter Java mit Applets**
3. Mischformen sind möglich und sinnvoll – einfache Logik zwecks Plausibilitäts- und Fehlerkontrolle auf Client, komplexe Datenbankaktionen auf Server -> Thin-Fat-Client mit **mit AJAX**

Webapplikationen mit Java

Grundkonzepte

A: Serverseitige Anwendung (Thin Client)

- Skriptlösung : Java Servlet in einem Servletcontainer wie Tomcat
- HTML-orientierte Lösung: Java Server Pages (siehe Vortrag)

B. Clientbasierte Applikation (Fat Client)

- Laden eines Java-Applets in den Browser
- Java Applet baut ggf. Verbindung zum Server zwecks Datenbankabfrage auf

C : Thin-Fat-Client mit Ajax

- - JavaScript kommuniziert per XML-Nachrichten mit Webserver und zeigt neue Inhalte ohne Neuladen der Seite an (Web 2.0)

Java – Servlets

- Servlets sind Java-Objekte, welche von javax.servlet.Servlet abgeleitet werden
- Servlets werden innerhalb eines Servlet-Containers (oder auch Servlet Engine) instanziert und gestartet
- Häufigste Verwendung als HTTPServlet, das auf Browser-Anfragen (Request) antwortet
- Servlets bekommen vom Client (dem Browser) Parameter im Request-Objekt übergeben
- Servlets erzeugen HTML-Code, der als Response zum Client zurückgeliefert wird
- Servlet-Container sind Applikations-Server, die die J2EE Spezifikation erfüllen. Bekannte Hersteller hierfür sind Bea (WebLogic), IBM (WebSphere), Apache (Tomcat, JBoss, Open-Source), Enterprise Server (Borland), ...

Java – Servlet – Beispiel

- Java-Programm zur Erzeugung von HTML-Ausgaben
- Beispiel erzeugt mit JBuilder-Assistenten für Java-Servlets :

```
public class Servlet1 extends HttpServlet {  // HTTP-Servlet stellt alle Basisfunkt.
  private static final String CONTENT_TYPE = "text/html";
  /**Globale Variablen initialisieren*/
  public void init() throws ServletException {
  }
  /**Die HTTP-Anforderung Get bearbeiten*/
  public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType(CONTENT_TYPE);
    PrintWriter out = response.getWriter();
    out.println("<html>");
    out.println("<head><title>Servlet1</title></head>");
    out.println("<body>");
    out.println("<p>Das Servlet hat ein GET empfangen. Das ist die Antwort.</p>");
    out.println("</body></html>");
  }
```

- Alle Requestparameter werden durch HttpServletRequest request übergeben
- Response definiert out.println() - übergibt Texte an den Ausgabestream des Servers

Java – Servlets - Methoden

Initialisierung

- Methode void init () - Server lädt und erzeugt Instanz des Servlets

Interaktion mit Clients (Behandlung von Anfragen)

Über Objekt HttpServlet

- void doGet (HttpServletRequest, HttpServletResponse)
  => zur Behandlung von GET-Request (URL-Eingabe)
- void doPost (HttpServletRequest, HttpServletResponse)
  => zur Behandlung von POST-Request (Formular-Actio-Button)

- void destroy () - Server entfernt Servlet-Instanz (i.d.R. erst beim Stoppen des Servers) => zum Aufräumen , z.B. zum Schliessen von Logfiles oder Datenverbindungen ....

Java – Servlets - Request-Objekt

Das Requestobjekt wird beim Aufruf über DOGEt / DoPOst übergeben und enthält alle Informationen über den Aufruf von Seiten des Servers

Konkrete Methoden / Attribute :

- String getParameter (String) => liefert den Wert eines konkreten Parameters.
- Enumeration getParameterNames () - liefert die Namen sämtlicher Parameter, die im request- Objekt mitgeliefert wurden.
- String[] getParameterValues (String) - Diese Methode ist für Parameter geeignet, die mehr als einen Wert enthalten können. Zurückgeliefert wird ein ganzes Array von Werten des benannten Parameters.

Java – Servlet- Formularverarbeitung

- DoPost verarbeitet Formulardaten aus request.getParameters()

```
/**Die HTTP-Anforderung Post bearbeiten*/
  public void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    response.setContentType(CONTENT_TYPE); PrintWriter out =
response.getWriter();
    out.println("<html>");   out.println("<head><title>Servlet1</title></head>");
    out.println("<body>");  out.println("<p>POST - Das ist die Antwort.</p>");

  out.println("Parameterliste: ");
    Enumeration par;
    par = request.getParameterNames();
    String parname = (String)par.nextElement();
    out.println("<BR>P:"+parname);
    while ( par.hasMoreElements() )
    { out.println("<BR>P: "+(String)par.nextElement()); }
    String parvalue = request.getParameter(parname);


    out.println("<B><BR>P:"+parname + " = " + parvalue+"</B>");
  out.println("</body></html>");
  }
```

Java – Server Pages (JSP)

Probleme beim Einsatz von Servlets

- Vermischung von Präsentation und Verarbeitungslogik durch gleichzeitige Programmierung in EINEM Code
- **HTML nur als Print-Ausgabe, damit Designer-Werkzeuge für HTML nicht verwendbar !**

Lösung : Java Server Pages (JSP)

- konzipiert für Web-Seiten mit dynamisch generiertem Inhalt
- Definition im HTML-Format mit eingebettetem Java-Code
- **wird zu Java-Servlet übersetzt !**

Folgen :

- Striktere Trennung von fachlicher Logik und Präsentation
- Änderungen in GUI unabhängig von Verarbeitungslogik
- Verwendung wiederverwendbarer Komponenten, z.B. mit Komponenten (z.B. JavaBeans)

Java-Server-Pages-Details

- Erstellung innerhalb eines statischen HTML-Dokumentes
- Einbettung von Java-Code über <% ... %>
- Deklarationen über <%! ... %>
- Die Objekte out und request stehen ohne Deklaration zur Verfügung.

Versteckte Kommentare (Hidden Comment)

- Syntax: <%-- Kommentar --%>
- Zum Dokumentieren des Quellkodes - wird von der JSP-Engine nicht verarbeitet und wird somit auch nicht zum Client gesendet

Ausgabe Kommentar (Output Comment)

- Syntax: <!-- Kommentar [<%= Ausdruck %>] --!>
- Wird von der JSP-Engine verarbeitet und das Ergebnis taucht in der gesendeten HTML-Quelldatei auf Der im Kommentar enthaltene Ausdruck wird dynamisch zur Laufzeit bei jedem Aufruf evaluiert

Java-Server-Pages-Beispiel

- Die Datei wird geparst und der Java-Code wird übersetzt !

```
<html><head><title>JSPTest</title></head>
<h1>Spiegel Java Server Page</h1>
<body> <form method="post">
Eingabe: <input type=text name=eingabe />
<br>Ausgabe:
<%! // eine Deklaration
String eingabe, ausgabe; %>
<% // hier steht nun Java-Code ...
eingabe = request.getParameter("eingabe");
if(eingabe != null) {
ausgabe = ServletTest.reverse(eingabe);
out.println(ausgabe);
}
// Ende des Java-Codes
%>
<input type=submit name=Submit />
</form></body></html>
```

Tomcat

- Das Programmpaket TOMCAT ist eine Servlet-Engine.

Diese Engine :

- verwaltet das Starten und die Parameterübergabe zwischen Webserver und Servlet.
- Bei einem Aufruf des Servlets wird dieses durch die TOMCAT-Engine aktiviert und bleibt anschließend aktiv.
- Die TOMCAT-Engine kann auch ein Servlet mehrfach ausführen und verfügt dazu über Multitasking-Fähigkeit

Tomcat kann in drei Modi arbeiten

- als einfacher Webserver (siehe Übung) auch mit eigener SSL-Verschlüsselung
- integriert in andere Webserver (allgemeiner Modul)
- Speziell angebunden an Apache mit Zusatzfunktionen

Tomcat in der Apache-Gesamtkonfiguration

- TOMCAT als Servlet-Engine an einem Apache-Server