Kopplungstechniken und Zugriffsschnittstellen

- Aufgaben und Anforderungen
- Techniken
  - Gateways, Programmierschnittstellen: CGI
  - Einbettung, HTML/SQL-Integration: PHP, ASP
  - Makroprogrammierung: IDC
  - Vergleich
- Zustandsrealisierung
- Java: Servlets und Server Pages

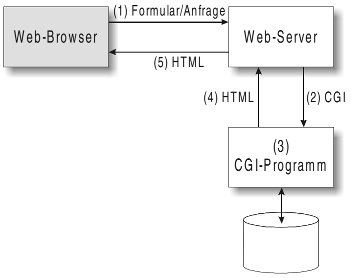
Aufgaben und Anfordeungen

- Aufruf „externer“ Programme mit Datenbankoperationen
- Parameterübergabe Client → Server → DB-Programm
- dynamische Generierung von Web-Dokumenten mit Ergebnissen von DB-Operationen
- Realisierung von Zuständen (Sitzungen, Transaktionen)

Common Gateway Interface (CGI)

- CGI: ursprüngliche Schnittstelle zwischen Web-Server und Programm zur Anbindung externer Quellen
- **Gateway**:
  - Umformung einer Informationsquelle in ein Web-Dokument
  - Programm/Skript zur Bearbeitung einer Dokumentenanforderung, das
    - HTML-Text, eine URL oder andere Daten liefert
    - in beliebiger Sprache implementiert ist

CGI: Prinzip



Parameterübergabe

- Übergabe von Parametern an CGI-Programme durch Browser
  - z.B. Inhalte von Formularelementen
- Methoden
  - GET: durch Anhängen an URL
    - für wenige Parameter geeignet (Längenbegrenzung)
    - kann direkt mit URL angegeben werden
    - Parameter für Nutzer sichtbar
  - POST: über Standardeingabe des CGI-Programms
    - für Nutzer nicht sichtbar
    - Parameter nicht Teil der URL

GET-Methode

- Parameteraufbau
  - Trennung von URL durch **`?`**
  - Trennzeichen **`&`**
  - Ersetzung von Leerzeichen durch **`+`**
  - Umlaute durch Escape-Notation **`%hexwert`**
- Beispiel:
  - Parameter:
    - feld1: "Eins"
    - feld2: "Zwei Drei"
  - URL:
    `http://host/cgi-prog?feld1=Eins&feld2=Zwei+Drei`

POST-Methode

- Parameteraufbau wie bei GET-Methode
- Übergabe als Standardeingabe
  - unbegrenzte Länge
  - geeignet für große Datenmengen
- Beispiel (Eingabedatei)
  `feld1=Eins&feld2=Zwei+Drei`

Kommunikation

- CGI-Programm → Web-Server
  - Erzeugung eines HTML-Dokumentes und Senden zur Standardausgabe
  - HTML-Dokument muß Content Type enthalten
  - Aufbau:
    ```
    Content-Type: text/html
    ```
    Leerzeile !!!!
    ```
    <html>
     ...
    </html>
    ```
  - Umlenkung durch Senden einer neuen URL
    ```
    Location: URL
    ```
    Leerzeile !!!!

Kommunikation

- Web-Server → CGI-Programm
  - Umgebungsvariablen, die vor Ausführung des Programms vom Server initialisiert werden

Variable	Bedeutung	Beispiel
REQUEST_METHOD	Request-Methode	GET
REMOTE_HOST	Host des Browsers	xxx.xx.xxx.xx
REMOTE_USER	Benutzername (nur bei Authentisierung)	
QUERY_STRING	Anfrageparameter	feld1=12&feld2=abc
CONTENT_LENGTH	Länge der Anfrageparameter	18
CONTENT_TYPE	Format der Anfrageparameter	application/x-www-form-urlencoded

CGI: Beispiel

- Bourne-Shell-Programm

```
#!/bin/sh
echo "Content-Type: text/html"
echo ""
echo "<html><body>Zugriff um = "
date
echo " von: "
echo $REMOTE_HOST
echo " mit Parametern: "
echo $QUERY_STRING
echo "</body></html>"
```

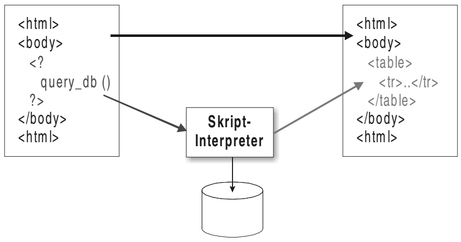

Weiterentwicklungen

- Probleme des CGI
  - Performance: jeder Aufruf =
    - Start eines neuen Prozesses
    - Aufbau einer neuen DB-Verbindung
  - Programm läuft unter UID des Web-Servers
- FastCGI
  - „persistente“ Prozesse
  - Interprozeßkommunikation zwischen Server und FastCGI-Prozeß
- Server-Erweiterungen
  - Erweiterung als Programmbibliothek, die zum Server dynamisch gebunden wird (NSAPI, ISAPI)

Einbettung

- Einbettung von Kommandos (z.B. Datenbankoperationen) in HTML-Dokument
  - Skriptsprachen wie Perl, Visual Basic, PHP, ...
- Kennzeichnung durch spezielle Tags (z.B. `<% ... %>`
- Verarbeitung
  1. Parsen des Dokumentes vor Auslieferung
  2. Intepretieren der Kommandos in Tags
  3. Ersetzen durch Ergebnis der Ausführung

Einbettung: Prinzip



Einbettung mit PHP

- PHP: Skriptsprache für server-seitige Einbettung in HTML
- Features
  - C-ähnliche Syntax, objektorientierte Konzepte
  - einfache Verarbeitung von Dokumentanforderung
  - dynamische Generierung von HTML-Dokumenten
  - Schnittstellen zu vielen Systemfunktionen und DBMS
- Prinzip

```
<html><body>
<?php echo "Hello World !" ?>
</body></html>
```


Einbettung mit PHP

- Einbindung externer Dateien (z.B. HTML-Vorlagen)
  - **require** *'dateiname'*
    Einfügen des Inhaltes der Datei *dateiname*
  - **include** *'dateiname'*
    Einfügen und Auswerten der Datei *dateiname*
- Verarbeitung von Formularen
  - Auswertung der Umgebungsvariablen durch PHP-Interpreter
  - Zugriff auf Werte der Felder: Variablen mit Elementnamen
  - Beispiel:
    HTML: `<input type="text" name="eingabe">`
    PHP: `$eingabe`

PHP-Beispiel

- Formular

```
<form method=POST
        action="/path/query.php3">
    <input type="text" name="autor">
    <input type="submit">
</form>
```

- PHP-Skript

```
<html><body>
<table border=1>
<tr><th>Autor</th><th>Titel</th><th>Preis<
/th></tr>
```


PHP-Beispiel

```
<html><body>
 <table border=1>
 <tr><th>Autor</th><th>Titel</th><th>Preis</th></tr>
 <?php
 if (isset($author)) {
   $query = "SELECT autor, titel, preis FROM " .
            "buch WHERE autor like='%$author%'";
   mysql_connect ($host, $user, $passwd);
   mysql_select_db ("my_db");
   $result = mysql_query ($query);
   while ($row = mysql_fetch_row ($result)) {
      echo "<tr><td>$row[0]</td><td>$row[1]</td>";
      echo "<td>$row[2]</td></tr>";
   }
 }?>
 </table></body></html>
```


Active Server Pages

- Server-Erweiterung für MS Internet Information Server
    - server-seitige Scripting-Umgebung
    - Framework für Anwendungsentwicklung
- Features
    - sprachunabhängig: Unterstützung von VBScript, JScript, Perl, ...
    - vordefinierte Objekte für häufig benötigte Funktionen

ASP: Einbettung von Skripten

- Prinzip ähnlich PHP
  - spezielle Tags: `<% ... %>`
  - Beispiel:
    `<html><body>Zeit: <%= Now %></body></html>`
- Angabe einer Skriptsprache
  - Standard: VBScript
  - Umschaltung mit SCRIPT-Element
  - Beispiel

```
<SCRIPT RUNAT=SERVER LANGUAGE=VBSCRIPT>
SUB basicFunc
 Response.Write ("VBScript !")
END SUB
</SCRIPT>
```


ASP: Built-In-Objekte

- Vordefinierte und instantiierte Objekte für jede ASP-Anwendung
  - **`Application`**: Informationen zur Anwendung (für alle Sitzungen)
  - **`Session`**: Informationen zur aktuellen Sitzung (z.B. Warenkorb)
  - **`Request`**: Dokumentanforderung (z.B. Formulardaten)
  - **`Response`**: Antwort (z.B. HTML-Ausgabe, Cookies, ...)
  - **`Server`**: Schnittstelle zum Server (DB-Zugriff, ...)
- Beispiel:
  `<P><%= Request.Form ("autor")%><P>`

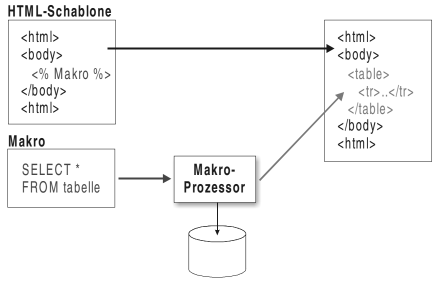
ASP: Installierbare Objekte

- Instantiierung bei Bedarf
- bieten zusätzliche Funktionalität
  - *Browser Capabilities*: Bestimmung der Eigenschaften des Browser
  - *File Access*: Funktionen zur Dateiarbeit
  - *Ad Rotator*: Werbebanner
  - *Content Linking*: konfigurierbare Link-Strukturen
  - *Database Access*: Zugriff auf Datenquellen (ActiveX Data Objects)

Makroprogrammierung

- Trennung von Dokument (Präsentation) und Datenbankoperationen (Logik)
  - HTML-Template mit Gerüst der Ergebnisseite
  - Makros mit SQL-Anfragen oder Änderungsoperationen
  - Verbindung über Platzhalter im HTML-Dokument
- Ablage in
  - verschiedenen Dokumenten (verbesserte Wiederverwendbarkeit)
  - in einem Dokument mit getrennten Bereichen

Makroprogrammierung



Internet Database Connector (IDC)

- Einfache Erweiterung für MS Internet Information Server auf Basis von ODBC
  - stark eingeschränkte Skriptsprache
  - kaum Anwendungslogik implementierbar
  - wenig Kontrolle über Ausgabe
  - einfach anwendbar
- Prinzip
  - idc-Datei mit DB-Operationen
  - htx-Datei mit HTML-Templates

IDC: Aufbau der idc-Datei

- beinhaltet
  - ODBC-Datenquelle (**`Datasource`**)
  - Verweis auf HTML-Template (**`Template`**)
  - Anfragen (**`SQLStatement`**)
- **`SQLStatement`** mit Anfrageparametern (Wert eines Formularelementes)

```
SQLStatement:
+ SELECT autor, titel, preis
+ FROM buch
+ WHERE autor=<%autor%>;
```


IDC: Aufbau der htx-Datei

- Vorlage für HTML-Seiten mit Anfrageergebnissen
- bestehend aus
  - Kopf-, Fußbereich (unabhängig vom Ergebnis)
  - und Detailbereich (in Abhängigkeit vom Ergebnis: Wiederholung für jedes Ergebnistupel)
- Kennzeichnung des Detailbereiches
  `<%begindetail%> <%enddetail%>`
- Kennzeichnung der Ergebnisdaten
  <%*attribut*%>

IDC: Aufbau der htx-Datei

- Beispiel:

```
<table border=1>
    <tr><th>Autor</th><th>Titel</th>
          <th>Preis</th></tr>
    <%begindetail%>
          <tr><td><%autor%></td>
          <td><%titel%></td>
          <td><%preis%></td></tr>
    <%enddetail%>
</table>
```

IDC: Kontrollstrukturen

- Unterstützung bedingter Ausführung durch `if...else...endif`
- spezielle Tags:
  `<%if` *bedingung* `%>` *HTML*
  [`<%else%>` *HTML]* `<%endif%>`
- Bedingungen mit
  - Operatoren `EQ, LT, GT, CONTAINS`
  - Variablen: Ergebnisdaten, Servervariablen, spezielle Variablen (`CurrentRecord, MaxRecords`)

IDC: Aufruf

- Aufruf der IDC-Skripte wie CGI-Programme als URL im ACTION-Attribut eines Formulars

```
<form method=post
   action="/shop/query.idc">
   ...
   <input type="text" name="autor">
   <input type="submit">
</form>
```


Vergleich und Bewertung

- **Programmierschnittstellen**
  - Vorteile
    - Performance
    - Codegröße
    - Optimierbarkeit für konkrete Anforderungen
  - Nachteile
    - unflexibel (u.U. Neuübersetzung bei Änderungen)
    - Aufwand:
      - neues Programm für jedes Problem
      - Generierung von HTML
    - Performance bei CGI

Vergleich und Bewertung

- **Einbettung**
  - Vorteile
    - Lesbarkeit durch Plazierung von SQL-Anweisungen an die spätere Position der Daten
    - Erstellung und Wartung mit HTML-Wekzeugen möglich
    - eine gemeinsame Datei
  - Nachteile
    - bei größeren Projekten unübersichtlich durch Mischung von HTML und SQL
    - Performanceprobleme durch Parsen und Interpretieren von eingebetteten Anweisungen

Vergleich und Bewertung

- Makroprogrammierung
  - Vorteile
    - Bearbeitung von HTML-Dateien mit HTML-Werkzeugen möglich
    - übersichtliche Spezifikation aller SQL-Anfragen
    - Wiederverwendbarkeit von Anfragen und HTML-Templates
    - schnelleres Parsen durch kleinere Dateien
  - Nachteile
    - schlechte Überschaubarkeit durch Verteilung auf mehrere Dateien
    - mehrere Dateizugriffe notwendig

Realisierung von Zuständen

- Problem:
  - Zustandslosigkeit von HTTP
  - im Normalfall: *Anforderung* =
    1. Aufbau der DB-Verbindung
    2. Ausführung der Anfrage
    3. Abbruch der Verbindung
  - keine Sitzung mit mehreren Verarbeitungsschritten möglich (Transaktionskonzept)
  - potentiell paralleler Zugriff vieler Nutzer

Realisierung von Zuständen

- Aufgabe:
  - server-seitige „Speicherung“ des Kontextes (Zustandes) des Clients → *Session-ID*
- Verfahren:
  - URL-Kodierung
  - Formularvariable
  - Cookies
  - HTTP-Authentisierung
- notwendig: Timeout-Mechanismen für Sitzungsende und Freigabe belegter Ressourcen

URL-Kodierung

- Übergabe der Session-ID als Teil der URL
- explizite Kodierung als Parameter (GET-Request)
  - Beispiel
    `http://bookshop.com/buy?session=42`
  - server-seitige Behandlung wie Formularvariable
- Kodierung in Pfad
  - Beispiel:
    `http://bookshop.com/42/daten.html`
  - bei relativer Adressierung kein weiterer Aufwand zur Generierung
- Vorteil: Browser-unabhängig
- Problem: Initialisierung

Formularvariable

- Speichern der Session-ID in einem (versteckten) Formularelement

```
<input type="hidden" name="session"
    value="42">
```

- Übergabe des Wertes mit HTTP-Request (GET, POST)
- Sitzungsmanagement
  - Auslesen des Feldwertes
  - Erzeugen eines neuen Feldes mit aktuellem Wert für Folgeseiten
- Vorteil: Browser-unabhängig
- Probleme:
  - Im HTML-Quelltext sichtbar
  - dynamische Generierung der Folgeseiten notwendig

Cookies

- Metainformation zu Web-Dokumenten
- Einbettung in HTTP-Header bei Übertragung des Dokumentes zum Browser
- Aufbau:
  `Set-Cookie:` *Name*`=`*Wert*`; expires=`*Datum*`;`
  `path=`*Pfad*`; domain=`*Domäne*
  - *Name*, *Wert* des Cookies
  - Gültigkeit
    - Zeit: *Datum*
    - Web-Server/Bereich: *Pfad*, *Domäne*

Cookies

- Beispiel
  `Set-Cookie: CustId="42"; Path="/bookshop"`
- Cookies werden vom Browser
  - persistent gehalten (bis zum Verfallszeitpunkt)
  - bei Anforderung eines Dokumentes aus dem jeweiligen Bereich zum Server übertragen
    `Cookie: CustId=42`
- Vorteil: automatische Unterstützung durch Browser
- Problem: abschaltbar durch Nutzer

Cookies mit PHP

- Setzen eines Cookies

```
int setcookie (string name, string value,
        int expire, string path,
        string domain, int secure);
```

- Beispiel:

```
setcookie ("CustId", "42", time () + 3600,
        "/bookshop");
```

- Lesen eines gesendeten Cookies
  - Cookie wird als Variable übergeben
    `$CustId`

HTTP-Authentisierung

- Nutzung der Authenisierungsmechanismen von HTTP
- Nutzer/Paßwort-Abfrage durch Browser
- Speichern der Nutzerinformation in der Umgebungsvariablen `REMOTE_USER`
- automatische Übermittlung durch Browser nach erfolgreicher Anmeldung
- Problem
  - erfordert vorherige Registrierung

Server-seitiges Java

- Nutzung von Java im (Web-)Server
- Zugriff auf Java-Technologien
  - DBMS-Schnittstellen
  - CORBA, RMI für verteilte Verarbeitung
- Techniken
  - Servlets
  - Java Server Pages
- Vorteile:
  - Plattformunabhängigkeit
  - Performance

Servlets: Überblick

- Servlet = Java-Objekt, das im Kontext eines Web-Servers abläuft
- Alternative zu CGI
  - verarbeitet Dokumentanforderungen
  - liefert Dokumente als Ergebnis
- Unterstützung von
  - Cookies
  - Session-Management
- Ablauf
  - direkt im Server (Java-WebServer)
  - in separatem Prozeß (Servlet-Engines)

Servlet-Engines

- Unterstützung der Spezifikationen von Sun (javax.servlet)
- Portabilität der Servlets gesichert
- Engines
  - spezielle Server
    - Java-Webserver (Sun)
    - Jigsaw (W3C)
  - Erweiterung exisitierender Server (Apache, IIS, Netscape Enterprise Server, ...)
    - JServ (Apache Group)
    - JRun (Live Software)

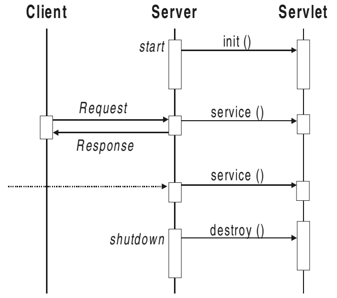
Servlet-Schnittstelle

- **`interface javax.servlet.Servlet`**
  - **`void init (ServletConfig config)`**
    Initialisierung des Servlets (z.B. Öffnen benötigter Dateien, Öffnen einer Datenbank-Verbindung, ...)
  - **`void destroy ()`**
    Aufräumarbeiten (z.B. DB-Verbindung schließen, Daten speichern)
  - **`void service (ServletRequest req, ServletResponse res)`**
    Bearbeitung der Anfrage (Parameterabfrage, Ergebnisgenerierung)

Servlets: Prinzip

- jedes Servlet muß Schnittstelle **`javax.servlet.Servlet`** implementieren
- Lebenszyklus
  1. Initialisierung durch Aufruf der **`init`**-Methode
     - beim Start der Servlet-Engine
     - bei der ersten Anforderung
  2. beliebig viele Aufrufe von **`service ()`** (Bearbeitung von Anforderungen)
  3. Beenden und Freigabe über **`destroy ()`**

Servlets: Prinzip



Basisklasse GenericServlet

- abstrakte Klasse javax.servlet.GenericServlet
- Basis für benutzerdefinierte Servlets
- protokollunabhängig (nicht nur HTTP !)
- implementiert wichtige Methoden
  - Schnittstelle zum Web-Server (Kontext, Konfiguration, Logging)
- Methode service () muß in abgeleiteten Servlet-Klassen überschrieben werden

Beispiel-Servlet

```
public class HelloServlet extends
  javax.servlet.GenericServlet {
     // Bearbeitung der Anforderung
     public void service (ServletRequest req,
                          ServletResponse res)
       throws ServletException, IOException {
        // MIME-Typ des Antwortdokuments
        res.setContentType ("text/html");
        // Ausgabestrom
        ServletOutputStream out =
          res.getOutputStream ();
        out.println ("Hello World !");
     }
  }
```

Ausführungsumgebung

- Zugriff auf Informationen und Dienste der Ausführungsumgebung (Servlet-Engine)
- Konfiguration `javax.servlet.ServletConfig`
  - Zugriff auf Intitialisierungsparameter der Konfigurationsdatei
    `String getInitParameter (String name)`
    `Enumeration getInitParameterNames ()`
  - Kontextobjekt
    `ServletContext getServletContext ()`

Ausführungsumgebung

- Kontext `javax.servlet.ServletContext`
  - *Zugriff auf Ressourcen:*
    `String getMimeType (String filename)`
    `String getRealPath (String path)`
    `RequestDispatcher`
    `getRequestDispatcher (String uri)`
  - *Log-Dienst:* Nachrichten in zentrale Log-Datei
    `void log (String msg)`
  - *Attribute:* von mehreren Servlets gemeinsam genutzte Objekte
    `void setAttribute (String name, Object o)`
    `Object getAttribute (String name)`

Anfrageobjekt: ServletRequest

- Zugriff auf Eigenschaften einer Anfrage
- Eigenschaften entsprechen CGI-Variablen + Parametern
- `javax.servlet.ServletRequest`
  - `int getContentLength ()`
    Länge der Anfragedaten
  - `String getContentType ()`
    MIME-Typ der Anfrage
  - `String getParameter (String name)`
    Wert des Parameters
  - `ServletInputStream getInputStream ()`
    Strom zum Lesen von Binärdaten

Antwortobjekt: ServletResponse

- Setzen von Eigenschaften bzw. Erzeugen des Ergebnisdokumentes
- **`javax.servlet.ServletResponse`**
  - **`void setContentLength (int len)`**
    Setzen der Ausgabelänge
  - **`void setContentType (String type)`**
    Setzen des MIME-Typs des Dokumentes
  - **`ServletOutputStream getOutputStream ()`**
    Strom zum Schreiben des Ergebnisses

Beispiel: Parameterverarbeitung

```
...
// MIME-Typ des Antwortdokuments
res.setContentType ("text/html");

// Ausgabestrom
ServletOutputStream out =
   res.getOutputStream ();
...
// Wert des Formularelementes "autor"
out.println ("autor = " +
   req.getParameter ("autor"));
...
```


Umlenkung von Anfragen

- **`javax.servlet.RequestDispatcher`**
  - Einbinden von anderen Ressourcen (z.B. HTML-Dateien)
    ```
    void include (ServletRequest req,
                  ServletResponse res)
    ```
    - Nutzung von Standard-Dokumentteilen (Header, Footer)
  - Weiterleiten von Anfragen an andere Servlets
    ```
    void forward (ServletRequest req,
                  ServletResponse res)
    ```
    - Trennung von Logik und Präsentation

Umlenkung von Anfragen: Beispiele

- Einbindung

```
res.setContentType ("text/html");
getServletContext ().getRequestDispatcher
    ("header.html").include (req, res);
printBody ();
getServletContext ().getRequestDispatcher
    ("footer.html").include (req, res);
```


Umlenkung von Anfragen: Beispiele

- Weiterleitung

```
res.setContentType ("text/html");
req.setAttribute ("result",
    getDBQueryResult ());
getServletContext ().getRequestDispatcher
    ("display-results").forward (req, res);
```


HTTP-Servlets

- Identifikation und Behandlung der HTTP-Befehle (GET, HEAD, POST, PUT, ...)
- Erweiterung von GenericServlet: **`javax.servlet.http.HttpServlet`**
  - implementiert **`service`**-Methode
  - Methoden für verschiedene HTTP-Requests
    - **`void doGet (HttpServletRequest req, HttpServletResponse res)`**
      Bearbeitung eines GET-Requests
    - **`void doPost (HttpServletRequest req, HttpServletResponse res)`**
      Bearbeitung eines POST-Requests

HTTP-Request und -Response

- **`HttpServletRequest`**:
  - Zugriff auf HTTP-Header
    `String getHeader (String name)`
    `Cookie[] getCookies ()`
  - Sessionmanagement
    `HttpSession getSession ()`
    `boolean isRequestedSessionIdValid ()`
- **`HttpServletResponse`**
  - Setzen von Header und Cookie
    `void setHeader (String name, String value)`
    `void addCookie (Cookie cookie)`
  - Senden von Fehlermeldungen
    `void sendError (int statusCode)`

Unterstützung von Cookies

- Java-Klasse: **`javax.servlet.http.Cookie`**
- Ermitteln von gesendeten Cookies aus HttpServletRequest
  `Cookie[] cookies = req.getCookies ();`
- Cookie-Eigenschaften
  - Name: **`String getName ()`**
  - Wert: **`String getValue ()`**
- Cookie erzeugen
  `Cookie mycookie = new Cookie ("id", "42");`
- Cookie setzen für Antwortobjekt HttpServletResponse
  `res.addCookie (mycookie);`

Session-Management

- Session (Sitzung): zusammenhängende Abfolge von Anforderungen und Antworten zwischen Client und Server
  - Cookies
  - URL-Kodierung
- Java-Klasse: **`javax.servlet.http.HttpSession`**
  - Ermitteln bzw. Erzeugen einer Session
    `HttpSession session = req.getSession (true);`
  - Session-Eigenschaften
    - Neu erzeugt: **`boolean isNew ()`**
    - Zeitpunkt des letzten Zugriffs: **`getLastAccessedTime ()`**
    - Erzeugungszeit: **`getCreationTime ()`**

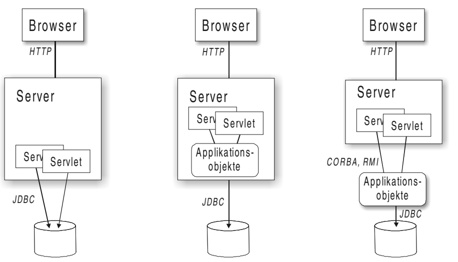
Session-Management

- Zuordnung von Java-Objekten zu Session
  - Warenkorb
  - Datenbankverbindung
  - Anfrageergebnis
- Zuweisen eines Objektes
  `void putValue (String name, Object value)`
- Ermitteln des zugewiesenen Objektes
  `Object getValue (String name)`
- Initialisierung bei Sitzungsbeginn
- Löschen bei Sitzungsende

Session-Management: Beispiel

```
// Sitzung ermitteln bzw. neu erzeugen
HttpSession session = req.getSession (true);
if (session.isNew ())
   // neuen Warenkorb anlegen
   session.putValue ("warenkorb",
                     new Warenkorb ());
else {
   Warenkorb korb = (Warenkorb)
        session.getValue ("warenkorb");
   // Produkte in Warenkorb ablegen
   ...
}
```


Architekturvarianten



Scripting mit Java: Java Server Pages

- Einbettung mit Java: Java Server Pages (JSP)
  - Java-Quelltext in HTML
  - Integration von Java-Komponenten (Beans)
- Prinzip
  - basierend auf Servlets
  - Compilieren des eingebetteten Codes: → Generierung eines Servlet
  - bei wiederholter Anforderung: → Nutzung des compilierten Servlets
- Vorteil: Performance

JSP: Überblick

- Zugriff auf
  - vollständigen Java-Sprachumfang
  - implizite Objekte (Requests, Sessions)
- Server-Page
  - Übersetzungseinheit: HTML + Scriptlets (eingebetteter Java-Code) + eingebundene Dateien (über **`include`**-Direktive)
- Markierung in HTML-Seite **`<% ... %>`**

JSP: Syntax

- Include-Direktiven

```
<%@ include file="relativeURL" %>
```

- Deklarationen (Variablen, Methoden)

```
<%! String printMessage (String s) {
      return "ERROR: " + s; } %>
<%! int var = 0; %>
```

- Expressions (Ausdrücke): Einfügen von Werten

```
<%= output %>
<%= Math.sqrt (2) %>
```

JSP: Syntax

- Scriptlets (Blöcke)

```
<% String output = "Hello World !"; %>
<% if (request.getParameter ("autor") ==
   null) { %>
   ... HTML ...
<% } else { %>
   ... HTML ...
<% } %>
```

- Page-Direktive

  `<%@ page` *Attributeliste* `%>`

JSP: Syntax

- Page-Direktive: Attribute
  - **`import`**: importierte Java-Pakete
  - **`session`**: Unterstützung von Sessionmanagement
  - **`errorPage`**: Verweis auf Seite für Fehlermeldung
  - **`contentType`**: MIME-Typ
- Weiterleitung

  `<jsp:forward page="relativeURL" />`

Einbindung von Komponenten

- Einbinden von Java-Beans (Komponenten)
  - wiederverwendbarer Softwarebaustein (Objekt, Menge von Objekten) mit Eigenschaften (Properties), die gesetzt bzw. ausgelesen und in HTML-Seite eingebunden werden können
- Syntax

  `<jsp:useBean id="`*instanzName*`"`
  `scope="`*bereich*`" class="`*klasse*`" />`

- Laden und Instantiieren einer Komponente der Klasse *klasse* mit Namen *instanzName*

Einbindung von Komponenten

- Scope: Gültigkeitsbereich
  - **page, request, session, application**
- Setzen von Eigenschaften einer bezeichneten Komponente
  <jsp:setProperty name="*instanzName***"**
  property="*attribut***" value="***wert***" />**
- Auslese von Eigenschaften einer bezeichneten Komponente
  <jsp:getProperty name="*instanzName***"**
  property="*attribut***" />**

Einbindung von Komponenten

- Beispiel: Kalenderkomponente

```
...
<jsp:useBean id="calendar" scope="page"
class="employee.Calendar" />
<h2>
Calendar of <jsp:getProperty
name="calendar" property="username" />
</h2>
```


Zusammenfassung

- Techniken des Zugriffs auf externe Datenquellen
  - Gateways, Server-Erweiterungen
  - Einbettung in HTML
  - Makroprogrammierung
- Auswahl abhängig von Anwendung (Aufwand, Verhältnis HTML/Code, Architektur)
- Java-Technologien Servlet-API und JSP als Basis für komplexere, integrierte Applikationsinfrastrukturen (Application Server)