SERVLETS

In diesem Kapitel:

- Architektonische Einordnung von Servlets
- HTML Support
  - *Inline HTML Generierung*
  - *Server Side Includes*
  - *Servlets als Ersatz für CGI Skripts*
- Installation von Servlets
  - *Temporäre versus permanente Servlets*
  - *Einsatz des servletrunners*
  - *Einsatz des Java Web Servers*
  - *Einsatz der J2EE Reference Implementation (Apache)*
- Servlet API
  - *Der Servlet Lebenszyklus*
    - Die `init()` Methode
    - Die `service()` Methode
    - Die `destroy()` Methode
    - Einfache Servlets
- Servlet Context
  - *Servlet Initialisierung*
  - *Server Context Informationen*
  - *Servlet Context während einem Service Request*
- Hilfsklassen
- HTTP Unterstützung
  - *Die GTTP GET Methode*
  - *Die HEAD Methode*
  - *Die POST Methode*
  - *HTTP Support Klassen*
- Zusammenfassung
  - *javax.servlet : Servlet Support*
  - *javax.servlet.http : HTTP Servlet Support*
- Servelt Beispiele
  - *Generierung von Inline Content*
  - *Verarbeitung von HTTP POST Requests*
  - *Cookies*
  - *Session Information verwalten*
  - *Datenbankverbindungen*
- Sicherheitsfragen
  - *Die Servlet Sandbox*
  - *Access Control Listen*
- Multithreading
- Neuerungen inder neusten Version

Servlets eine Einführung

1.1. Einleitung

Servlets sind Java Programme, mit denen Web Server erweitert werden können. Die Technologie verhält sich ähnlich wie die Applets auf der Client-Seite. Servlets unterstützen ein Request / Response Entwurfsmuster, wie es typischerweise von Web Servern und Web Services unterstützt wird. In einem solchen Modell sendet ein Client einen Request in Form einer Message an den Server und der Server antwortet mit einer Antwort Message.

Servlets sind Teil der J2EE, der Java Enterprise Edition. Sie können die Servlet Klassen, die Sie für die Entwicklung eigener Servlets benötigen, entweder als J2EE SDK oder als JSDK (altes Java Servlet Developer Kit) oder einfach die aktuellen Klassenarchive (`servlet.jar`) direkt von Sun herunterladen.

Diese Klassen implementieren das Java Servlet API, werden also benötigt, um Servlets zu implementieren. Mit Servlets können Sie unterschiedliche Funktionalitäten implementieren, beispielsweise das Bearbeiten von HTML Formularen, Middle-Tiers, um Client Applikationen mit Datenbanken zu verbinden, Tunneling durch Firewalls, Einsatz als Ersatz für CGI Skripts und vieles mehr.

Das Java Servlet API basiert selber auf unterschiedlichen Java Technologien und Erweiterungen (javax Packages).

In dieser Kurseinheit lernen Sie

- wie Sie das Java Servlet API einsetzen können
- eigene Servlets zu entwickeln
- Servlets in einem Web Server zu installieren
- wie man auf die Parameter der HTML Formulare zugreifen kann.
- wie man Servlets als Middle Tier einsetzen kann.

Servlets sind Java Komponenten, welche in Java fähige Web Server eungebaut werden können und damit die Funktionalität des Servers erweitern.

Mögliche Services sind beispielsweise:

- neue Funktionalitäten
- Änderung des Contents von Web Seiten zur Laufzeit
- Änderungen der Web Seiten Präsentation
- Unterstützung neuer Protokolle

Servlets arbeiten, wie oben bereits erwähnt, im Rahmen des Request / Response Patterns. In diesem Modell sendet ein Client eine Request Message an den Server und der Server beantwortet diese Anfrage mit einer Antwort Message.

Die Requests können basieren auf:

```
  HTTP
  URL,
  FTP,
  URL,
..eigenen Protokollen
```

Eine Anfrage und deren Beantwortung charakterisieren den Zustand des Clients und des Servers zum Zeitpunkt der Anfrage. In der Regel wird der Zustand zwischen zwei Anfragen nicht gespeichert. Aber die Session Information kann mit Servlets beschrieben und gespeichert werden.

Das Java Servlet Interface enthält andere Java Interfaces und beschreibt die Verbindung zwischen dem Client und dem Server vollständig. Das Servlet API ist als Erweiterung des JDKs definiert. Die javax Packages der Servlets bestehen aus:

- javax.servlet
- javax.servlet.http

Servlets sind eine mächtige Ergänzung der Java Umgebung. Servlets ind sicher, zuverlässig und bestehen aus 100% Java. Weil Servlets Plug-Ins für bestehende Web Server sind, kann man die bereits vorhandene Server Technologie optimal nutzen. Der Server hält die Netzwerkverbindung aufrecht, allfällige Protokollumwandlungen, das Laden der Klassen und vieles mehr. Dies braucht alles nicht neu erfunden zu werden. Und weil Servlets den Middle-Tier repräsentieren, können sie einiges an Flexibilität liefern.

In dieser Kurseinheit lernen Sie nicht alle Features und mögliche Einsatzgebiete der Servlets kennen. Aber nach dem Durcharbeiten der kurseinheit sollten Sie in der Lage sein, selber Servlets zu entwickeln und in Web Server einzubauen und zu nutzen.

1.1.1. Welche Rolle spielen Servlets in der Gesamtarchitektur?

Weil Servlets mächtig und flexibel sind, können sie eine wichtige Rolle in einer Systemarchitektur spielen. Servlets sind am Besten für Arbeiten / Aufgaben des Middle Tiers geeignet, beispielsweise als Proxys, oder zur Implementation neuer Protokolle. Middle Tiers sind Software Komponenten, die zwischen Thin Clients wie Web Browsern und eventuell massiven Datenquellen / Datenbanken vermitteln.

1.1.1.1. Middle-Tier Prozesse

In vielen Systemen dienen Middle Tier Server als Verbindung zwischen Clients und Backend Services. Middle Tiers können oft die Datenverarbeitung beim Client entlasten, also zu Thin Clients führen und dem Server seine Hauptaufgabe zurück geben (besipielsweise die effiziente Datenverwaltung).

Ein Vorteil einer 3-Tier Lösung ist die Vereinfachung des Verbindungsmanagements. Servlets können mehrere hundert Verbindungen verwalten und damit teure Datenbankserver entlasten.

Typische Middle Tier Funktionen sind:

- überprüfen der Business Rules
- Transaktion Management
- Serverfarmverwaltung
- Unterstützung unterschiedlicher Client Typen (HTML, WML, Java,...)

1.1.1.2. Proxy Servers

Falls Sie Applets einsetzen, können Sie Servlets auch als Proxys für die Applets einsetzen. Dies kann auf der einen Seite die Applikationssicherheit erhöhen, auf der anderen Seite den Server entlasten.

Falls Applets eine Datenbankverbindung benötigen, kann ein Servlet als Verbindung eingesetzt werden.

1.1.1.3. Protokoll Support

Das Servlet API stellt eine enge Verbindung zwischen einem Server und den Servlets her. Dies gestattet es Ihnen, neue Protokolle zu definieren. Aus dem API werden Sie sehen, wie HTTP Support eingebaut wurde. Das HTTP Protokoll kann also quasi als Prototyp für eigene Protokolle dienen. Einzige Anforderung an das Protokoll ist, dass es das Rquest/Response Pattern befolgt.

Hier einige Beispiele für solche möglichen Protokolle:

- SMTP
- POP
- FTP

Zur Zeit unterstützen verschiedene Web Server Servlets, zum Teil direkt, zum Teil mit Hilfe von Drittprodukten. Beispiele sind der Java Web Server von Sun und Apache. Aber auch Applikationsserver wie beispielsweise die J2EE Referenzimplementation unterstützen Servlets.

Da HTTP zur Zeit sehr beliebt ist, wurde dieses Protokoll speziell unterstützt.

1.1.2. HTML Support

HTML kann sehr sinnvoll zur Darstellung von Informationen eingesetzt werden, weil es sehr einfach ist, sehr leicht zu implementieren und leicht zu verstehen. Daher ist es naheliegend, dass Servlets in diesem Umfeld eine grosse Rolle spielen.

Daher wurde ein spezielles Package, `javax.servlet.http`, speziell für den HTTP Support durch Servlets entwickelt. Heute gibt es kaum mehr Web Sites ohne Personalisierung / Individualisierung der Inhalte (myWebSite...). Diese Anpassungen reichen von sehr einfach bis zu sehr komplex. Daher ist es sicher nötig spezielle Tools für diese Aufgabe zur Verfügung zu haben.

1.1.2.1. Inline HTML Generierung

Einige Web Server wie beispielsweise der Java Web Server (JWS) von Sun, gestatten es Servlet Tags direkt in die HTML Seiten einzubetten. Diese Technik ist aber nicht sehr üblich und nach dem "Tod" des Java Web Servers von Sun auch kaum mehr ein grosses Thema. Falls Sie direkt in HTML Seiten Java verwenden möchten, sollten Sie beispielsweise die Kurseinheit über Java Server Pages durcharbeiten.

1.1.2.2. Server-Side Includes

Diese Technik, Programmcode direkt in die HTML Seiten einzufügen, bezeichnet man als *server-side includes* (SSI). Mit SSL kann eine HTML Seite die unterschiedlichsten Befehle und Java Konstrukte enthalten. In der Regel müssen die Tags aber speziell gekennzeichnet sein, beispielsweise: <!--#include virtual="/includes/page.html"-->

Damit kann der Web Server feststellen, dass dieser Tag eine spezielle Behandlung benötigt und diesen teil der Web Seite an eine spezielle Serverkomponente weiterleiten.

Servlets können diese Tags auch verarbeiten, wenn Sie die nötigen Programme dafür bereitstellen! Sie könnten also mit Servlets eigene SSLs definieren und implementieren.

1.1.2.3. Ersetzen von CGI Skripts

Servlets können direkt als Ersatz für Common Gateway Interfaces (CGIs) eingesetzt werden. Ein HTTP Servlet ist ein direkter Ersatz für Common Gateway Interfaces (CGI) Skripts. HTTP Servlets werden mitels URLs eingesetzt:

```
http://localhost/servlet/DateTimeServlet
```

Dieses Servlet "hört" auf Anfragen und antwortet darauf, indem das Servlet eine HTML Seite an den Browser sendet. Allerdings beschränkt sich die Aufgabe der Servlets nicht nur auf das generieren einer Web Seite. Die obige Aufgabe könnten Sie mit JavaScript in einer Zeile lösen, schauen Sie auf der Kurs Web Seite nach: dort wird es mit JavaScript gemacht (wobei einige leute in ihrem Browser JavaScript aus Sicherheitsgründen ausschalten).

Das Servlet könnte auch Daten aus einer Datenbank lesen, aufbereiten und in eine HTML Tabelle einfügen, oder ein Mailsystem abfragen und die emails als Tabelle (wie bei Bluemail und ähnlichen Systemen) anzeigen.

Sie könnten aber auch alle anderen Java Technologien aus dem Servlet heraus einsetzen: Sockets, RMI, CORBA, JMS, ...

1.1.3. Installieren der Servlets

Servlets werden nicht wie andere Java Programme gestartet. Servlets erweitern Web Server. Daher ist es naheliegend, dass ein Servlet einen anderen Weg geht:

1. zuerst muss das Servlet in einem Server installiert werden. Dazu muss dieser Server über die Fähigkeit verfügen, Servlets überhaupt auszuführen.

2. dann kann ein Servlet Dienst von einem Client angefordert werden.

Wie bereits erwähnt existieren verschiedene Web Server, welche Servlets unterstützen. Zudem könen Sie die gängigen Server mit Servlet Engines nachrüsten. Für unsere Zwecke werden wir entweder J2EE, Apache / Jakarta oder den Servletrunner aus dem alten Java Servlet Development Kit (JSDK) oder den Java Web Server (JWS) einsetzen.

1.1.3.1. Temporäre versus Permanente Servlets

Servlets kann man entweder für jeden einzelnen Client Request starten und wieder stoppen. Alternativ kann man auch die Servlets beim Hochfahren des Servers starten und beim Herunterfahren des Servers wieder herunterfahren. Temporäre Servlets werden also auf Anfrage gestartet und bieten eine gute Variante, die Systemressourcen zu optimieren. Die permanenten Servlets werden mit den Server Start geladen und leben solange wie der Server läuft. Dies bietet sich an für Servlets, welche sehr oft gebarucht werden oder Routineaufgaben auf dem Server erledigen müssen.

Servlets werden als permanente Erweiterung des Servers installiert. Aber sie benötigen oft sehr viel Ressourcen und Zeit beim Starten, beispielsweise um eine Datenbankverbindung aufzubauen, oder falls sie eine RMI oder eine CORBA Verbindung aufbauen.

Ob ein Servlet temporär oder permanent ist, hängt nicht vom Servlet selber, sondern nur von der Installation im Server ab. Da die Servlets beim Starten des Servers geladen werden, ist es möglich gleich zu diesem Zeitpunkt zeitaufwendige Arbeiten zu erledigen.

Beispielsweise könnte einServlet sofort nach dem Verbindungsaufbau durch den Client eine Verbindung zu einem remote Rechner aufbauen, dort einloggen und Daten inklusive Kommandos zwischen dem Client Browser und dem remote Host hin und her senden und so dem Client zu einer remote Session verhelfen.

1.1.3.2. Einsatz von Servletrunner

Obschon der Servletrunner veraltet ist, kann er sehr gut für das Einarbeiten eingesetzt werden, da Sie nicht erst viele andere Software Komponenten konfigurieren müssen.

Einzige Änderung, die ich im Skript / Batch für das Starten des Servers gemacht habe:

`Start Java` ersetzen durch `%JAVA_HOME%\bin\java`

Die restlichen Einstellungen, wie beispielsweise der Einbezug der servlet.jar Datei, funktionieren ohne Änderung. Ich habe das JSDK auf D: im Root installiert

```
D:\jsdk2.1>
```

Nachdem Sie nun in etwa wissen, wie man das JSDK aufsetzen muss, können Sie sich langsam aber sicher an die erste Aufgabe machen.

Damit Sie nicht unnötige Konfigurationsprobleme haben, sollten Sie Ihre Servlets (Class Dateien) und die HTML Seiten in die Verzeichnisse des JSDK's kopieren. Später können Sie immer noch die Konfiguration ändern.

1.1.3.3. Übung

Installieren Sie das Java Servlet Development Kit und testen Sie Ihre Installation. beachten Sie dabei die oben besprochenen Anpassungen.

Nachdem Sie die Installation abgeschlossen haben, können Sie sie mit dem Browser testen.

```
http://localhost:8080/
```

Weitere Hinweise und Screen Shots finden Sie im Übungsteil.

1.1.3.4. Servlets und der Java Web Server

Sun's Java Web Server (JWS) ist eine vollständige Implementation des HTTP Protokolls. Sun hat die Unterstützung allerdings eingestellt und auf Tomcat gewechselt. Alle Programmquellen gingen an Apache über. Da dieser Web Server mit GUI recht leicht zu bedienen ist, bietet er sich für den Kurs an, obschon er veraltet und nicht sehr stabil ist!

Sie können diesen Web Server auch als Service installieren: dies ist zwar praktisch, aber für das Testen von Servlets eher störend, da Sie beim Testen der Servlets dauernd den Web Server hoch und runter fahren müssen, um die Servlets sauber aus dem Speicher zu entfernen.

Servlet können Sie installieren, indem Sie diese ins Verzeichnis `%JAVAWS_HOME%\servlets` kopieren. JWS erkennt Servlets, die sich dort befinden (eventuell in einem Subverzeichnis, falls Sie ein Package verwenden). Sie können die Servlets auch administrieren, mit dem Administrator Applet für den JWS (`Admin.bat` im `%JavaWS_HOME%` Verzeichnis). Dort haben Sie die Möglichkeit die Servlets zu konfigurieren und permanent zu laden, falls dies erwünscht ist.

Den Web Server können Sie mit Hilfe des Admin Tools hoch- und runterfahren.

1.1.3.5. Übung

Servlets mit dem JWS nutzen.

1.1.4. Servlet API

Das Java Servlet API definiert Interfaces zwischen Servlets und Servern. Dieses API besteht aus zwei Packages, welche das JDK erweitern, als `javax`:

- Package `javax.servlet`
- Package `javax.servlet.http`

Das API liefert Support in vier Kategorien:

- Servlet Lebenszyklus Management
- Zugriff auf Servlet Kontext
- Hilfsklassen
- HTTP spezifische Unterstützungsklassen

1.1.4.1. Der Servlet Life Cycle

Servlets laufen auf Web Server Plattformen, als Teil des selben Prozesses wie der Web Server selber. Der Web Server ist verantwortlich dafür, dass Servlets korrekt instanziert und wieder eliminiert werden. Ein Web Server kommuniziert mit dem Servlet über ein einfaches Interface, `javax.servlet.Servlet`. Dieses Interface besteht aus drei Hauptmethoden:

- `init()`
- `service()`
- `destroy()`

und zwei Hilfsmethoden:

- getServletConfig()
- `getServletInfo()`

Falls Sie die Applet Interfaces kennen, werden Sie eine bestimmte Ähnlichkeit zu den Java Applet Interfaces feststellen. Diese Ähnlichkeit ist naheliegend: das Applet kommuniziert mit einem Web *Browser*, das Servlet mit einem Web *Server*, aber die Aufgaben sind ähnlich, somit auch die Kommunikation.

Die init() Methode

Wenn ein Servlet zum ersten Mal geladen wird, wird die `init()` Methode des Servlets ausgeführt. Damit kann man alle Initialisierungen, wie öffnen von Dateien, Verbindungsaufbau zu einer Datenbank, Einloggen in einen Server und vieles mehr erledigen. Falls das Servlet permanent geladen wird, geschieht die Initialisierung gelich beim Starten des Web Servers. Sonst geschieht die Initialisierung sobald die erste Client Anfrage für dieses Servlet eintrifft.

Die `init()` Methode wird ausgeführt und beendet, bevor irgendwelche andere Anfragen an das Servlet herangetragen werden können, wie etwa ein Aufruf der `service()` Methode. Die `init()` Methode wird auch nur einmal ausgeführt. Falls Sie die Methode ausführen wollen, müssen Sie das Servlet entladen und wieder neu laden.

Die `init()` Methode akzeptiert als Argument eine Referenz auf ein `ServletConfig` Objekt, welches Initialisierungsargumente für das Servlet enthält. Dieses Objekt besitzt eine Methode `getServletContext()`, welche ein `ServletContext()` Objekt liefert. Dieses enthält Informationen über die Servlet Umgebung.

Die service() Methode

Die `service()` Methode ist der Kern des Servlets. Jede Request Message eines Clients liefert einen einfachen Aufruf der Servlet `service()` Methode. Die service() Methode liest den Request und produziert die passende Antwort Message aus den zwei Parametern:

- ein `ServletRequest` Objekt mit Daten für den Client. Die Daten bestehen aus Namens/Wert Paaren von Parametern und einem `InputStream`. Mit Hilfe mehrere Methoden kann man auf die Parameterinformationen des Client zugreifen. Den `InputStream` erhält man mit der Methode `getInputStream()`. Diese Methode liefert einen ServletInputStream, mit dem weitere Daten vom Client erhalten werden können. Falls Sie zeichenorientierte Daten anstelle von byteorientierten Daten verarbeiten möchten, können Sie auch einen `BufferedReader` erhalten, mit der Methode `getReader()`.
- ein `ServletResponse` Objekt repräsentiert die Antwort des Servlets an den Client. Als erstes muss dabei die Methode `setContentType()` aufgerufen werden, um den MIME Type korrekt zu setzen. Dann kann man die Methoden `getOutputStream()` oder `getWriter()` einsetzen, um einen `ServletOutputStream` oder `PrintWriter` Objekt zu erhalten, um Daten an den Client zurück zu senden.

Offensichtlich gibt es zwei Möglichkeiten, um als Client Informationen an ein Servlet zu senden:
1) die erste Variante besteht darin, Parameter Werte zu senden.
2) die zweite Variante besteht darin, die Informationen über den InputStream / Reader zu senden.

Parameter Werte kann man auch in eine URL einbetten. Wie dies gemacht wird, sehen wir weiter unten. Wie die Parameterwerte vom Servlet gelesen werden, sehen Sie auch später.

Die Aufgabe der `service()` Methode ist konzeptionell einfach - sie kreiert eine Antwort für jeden Client Request, der vom Host Server an sie gesendet wird. Aber es ist wichtig zu erkennen, dass mehrere Anfragen gleichzeitig bearbeitet werden können. Das impliziert, dass Sie darauf achten müssen, Ihre Anwendung Thread-Safe zu bauen, beispielsweise bei Zugriffen auf Dateien oder Datenbanken. Wie man Servlets Thread-Safe macht, werden wir noch genauer ansehen.

Die destroy() Methode

Die `destroy()` Methode wird aufgerufen, um dem Servlet Gelegenheit zu geben, alle Ressourcen aufzuräumen (offene Dateien, Datenbankzugriffe und ähnliches) bevor das Servlet entladen wird. Falls keine Aufräumarbeiten nötig sind, kann die Methode leer bleiben.

Der Server wartet mit dem Aufruf der `destroy()` Methode bis entweder alle Service Aufrufe erledigt sind, oder bis eine bestimmte Zeitdauer abgelaufen ist. Das heisst aber, dass im

Extremfall die `destroy()` Methode aufgerufen wird während andere längerlaufende `service()` Methoden immer noch am Laufen sind. Auch hier liegt es an Ihnen dafür zu sorgen, dass die destroy() Methode keine Ressourcen schliesst oder vernichtet, bevor alle `service()` Methoden beendet sind.

Servlet Beispiel

Sie haben bereits die Übungen zum HelloWorld Servlet durchgearbeitet. Daher sollte das folgende Servlet eigentlich nichts neues bringen. Das Servlet implementiert das gesamte API und liefert eine statische HTML Seite.

```
import java.io.*;
import javax.servlet.*;
public ServletBeispiel implements Servlet {
  private ServletConfig config;

  public void init (ServletConfig config)
    throws ServletException {
    this.config = config;
  }

  public void destroy() {} // da passiert nichts

  public ServletConfig getServletConfig() {
    return config;
  }

  public String getServletInfo() {
    return "Ein einfaches Servlet";
  }

  public void service (ServletRequest req,
    ServletResponse res
  ) throws ServletException, IOException  {
    res.setContentType( "text/html" );
    PrintWriter out = res.getWriter();
    out.println( "<html>" );
    out.println( "<head> );
    out.println( "<title>Ein einfaches Servlet</title>" );
    out.println( "</head>" );
    out.println( "<body>" );
    out.println( "<h1>Wie geht's so....</h1>" );
    out.println( "</body>" );
    out.println( "</html>" );
    out.close();
  }
}
```

Servlet Context

Ein Servlet lebt und stribt innerhalb der Grenzen des Server Prozesses. Um seine Umgebung besser verstehen zu können, kann ein Servlet Informationen über seine Umgebung bestimmen. Die Servlet Initialisierungsinformationen sind während dem Starten des Servlets erhältlich. Informationen über den Server sind jederzeit erhältlich und jeder Service Request kann sich diese Informationen zu Nutze machen.

Servlet Initialisierungs- Information

Initialisierungsinformation wird an das Servlet mit Hilfe des `ServletConfig` Parameter der `init()` Methode weitergereicht. Jeder Web Server stellt auf seine spezifische Art und Weise die Initialisierungsinformationen dem Servlet zur Verfügung.

Betrachten wir ein Beispiel mit dem Java Web Server (JWS) und einem Servlet, welches Datum und Uhrzeit liefern kann.

Wir könnten folgende Servlet Properties Datei kreieren:

```
servlet.dateprinter.code=DatePrinterServlet
servlet.dateprinter.timezone=PST
```

oder die selbe Information über ein GUI eingeben.

Die Zeitzonen- Informationen würden folgendermassen bestimmt:

```
String timezone;
public void init(ServletConfig config) {
  timeZone = config.getInitParameter("timezone");
}
```

Sie können auch alle Initialisierungsparameter, die dem Servlet zur Verfügung stehen, auflisten lassen: `getInitParameterNames()`.

Server Context Informationen

Server Context Informationen sind jederzeit über das `ServletContext` Objekt verfügbar. Ein Servlet kann dieses Objekt mit der Methode `getServletContext()` angewandt auf das Objekt `ServletConfig`, bestimmen. Die Informationen wurden während der Initialisierungsphase des Servlets festgelegt. Falls Sie die Information in der `init()` Methode bestimmen, sollten Sie diese in einer privaten Variable abspeichern, also nicht publik machen. Das `ServletContext` Interface definiert verschiedene Methoden:

`getAttribute()`	Serverspezifische Informationen, als Name / Wert Paare.
`getMimeType()`	liefert den MIME Type einer Datei.
`getRealPath()`	Liefert einen Relativpfad neu als Pfad zur Doc Base.
`getServerInfo()`	Name und Version des Netzwerkdienstes.
`getServlet()`	Liefert ein Servlet zum Namen, falls vorhanden.
`getServletNames()`	Liefert eine Liste mit den Namen der Servlets im Namensraum.
`log()`	Schreibt Loginfos ins Serverlog in dessen Format.

Das folgende Beispiel zeigt, wie ein Servlet Informationen ins Server Log schreibt:

```
private ServletConfig config;
public void init(ServletConfig config) {
  // zwischenspeichern der Informationen
  this.config = config;
  ServletContext sc = config.getServletContext();
  sc.log( "Erfolgreich gestartet!" );
}
```

Servlet Context während einem Service Request

Jeder Service Request kann Informationen in der Form von Namen/Werte Paaren enthalten. Diese Informationen sind im `ServletRequest` Objekt enthalten, welches an die `service()` Methode weitergereicht wird.

Das folgende Beispiel zeigt, wie diese Informationen erhalten werden können:

```
BufferedReader reader;
  String         param1;
  String         param2;
  public void service (
      ServletRequest  req,
      ServletResponse res) {

    reader = req.getReader();
    param1 = req.getParameter("DerErste");
    param2 = req.getParameter("DerZweite");
  }
```

Die folgende Tabelle enthält eine Liste der Methoden, die eingesetzt werden können:

`getAttribute()`	liefert den Wert eines benannten.
`getContentLength()`	Grösse des Requests, falls bekannt.
`getContentType()`	MIME Type des Message Body.
`getInputStream()`	InputStream für binäre Daten aus dem Body des Requests.
`getParameterNames()`	Liste aller Namen der Parameter.
`getParameterValues()`	Werte zu einem benannten Parameter.
`getProtocol()`	Protokoll: <protocol>/<major version>.<minor version>.
`getReader()`	BufferedReader zum Lesen von Text aus dem Body.
`getRealPath()`	Aktueller Pfad zu einem virtuellen Pfad (relativen Pfad).
`getRemoteAddr()`	IP Adresse des Clients.
`getRemoteHost()`	Host Name des Clients.
`getScheme()`	Benutztes Schema in der URL (https, http, ftp, etc.).
`getServerName()`	Name des Host Servers, der den Request empfangen hat.
`getServerPort()`	Port an dem der Request empfangen wurde.

Und nun schauen wir uns ein einem Beispiel das Ganze genauer an.

1.1.4.2. Übung - Zugriff auf Servlet Parameter

Arbeiten Sie jetzt die Übung zu den Servlet Parametern durch.

1.1.4.3. Hilfsklassen

Zum Servlet API existieren verschiedene Hilfsklassen. Die erste ist ein Interface, `javax.servlet.SingleThreadModel`. Mit diesem Interface kann man leicht einfache Servlets schreiben. Falls ein Servlet diese Marker Interface mplementiert, weiss der Server, dass die `service()` Methode nicht aufgerufen werden muss: alle Requests werden in einem einzigen Thread angehandelt.Nachteil dieses Interfaces ist die Performance. Diesen Fragen werden wir uns noch zuwenden.

Das Servlet API definiert auch zwei Exceptions:
1) `javax.servlet.ServletException` beschreibt allgemeine Ausnahmen in enem Servlet und informiert den Server über das Problem.
2) `javax.servlet.UnavailableException` zeigt an, dass ein Servlet nicht vefügbar ist.

Es gibt zwei Stufen der Nichtverfügbarkeit von Servlets:

- **Permanent**
  Das Servlet ist nicht verfügbar, bis der Administrator etwas unternimmt. Das Servlet sollte in diesem Fall einen Logfile Eintrag generieren.

- **Temporär**
  Bei einem Servlet ist temporär ein Problem aufgetreten, wie beispielsweise zuwenig Diskspace, Serverausfall .... Das Problem kann aber selbständig korrigiert werden und benötigt nicht unbedingt Unterstützung durch den Operator

HTTP Support

Servlets, welche das HTTP Protokoll benutzen, sind eigentlich die Regel. Daher sollte es nicht überraschen, dass für solche Servlets spezielle Klassen und Methoden zur Verfügung stehen. Diese Klassen sind im Package `javax.servlet.http` zusammengefasst. Bevor wir uns damit befassen, schauen wir uns das HTTP Protokoll genauer an:

HTTP steht für HyperText Transfer Protocol. Das Protokoll ist das von Web Browsern üblicherweise benutzte Protokoll. Es definiert ein Set von textbasierten Messages, die *HTTP Methoden*. (*HTTP Methoden* sind Message Requests, welche einen bestimmten Antworttyp verlangen). HTTP Methoden sind:
- GET
- HEAD
- POST
- PUT
- DELETE
- TRACE
- CONNECT
- OPTIONS

Hier benutzen wir lediglich GET, HEAD, and POST.

Die HTTP GET Methode

Die HTTP GET Methode holt Informationen vom Server. Diese Information kann eine Datei, die Ausgabe eines Serverdevices oder eines Programms (Servlet, CGI Skript) sein.

Ein GET Request sihet folgendermassen aus:

```
GET URL <http version>
Host: <target host>
```

Hier ein konkretes Beispiel:

```
GET / HTTP/1.1
Connection: Keep-Alive
User-Agent: Mozilla/4.0 (
 compatible;
 MSIE 4.01;
 Windows NT)
Host: www.magelang.com
Accept: image/gif, image/x-xbitmap,
 image/jpeg, image/pjpeg
```

Die meisten Server gestatten den URL basierten Zugriff auf Servlets mittels einer URL, welche mit `/servlet/` startet.

Hier ein Beispiel:

```
GET /servlet/MeinServlet?name=Simpson&
  company=Joller-Voss HTTP/1.1
Connection: Keep-Alive
User-Agent: Mozilla/4.0 (
 compatible;
 MSIE 4.01;
 Windows NT)
Host: www.joller-voss.ch
Accept: image/gif, image/x-xbitmap,
 image/jpeg, image/pjpeg
```

Die URL, welche in diesem GET Befehl eingesetzt wird, verwendet zwei Parameter: Name und Company. Jeder Parameter besitzt einen Namen und einen Wert. Parameter werden durch ein & getrennt. Die Parameter selbst werden der eigentlichen URL nach einem ? hinten angefügt.

Falls irgendwo ein %20 steht, bedeutet dies, dass ein Leerzeichen im Textstring vorhanden ist.
HTTP GET Requests haben eine klare Einschränkung: die meisten Server schränken die Anzahl Bytes, die in einem Request verwendet werden können, recht eng ein (einige hundert Bytes). Falls Sie mehr Daten übermitteln wollen, können Sie auf die POST Methode ausweichen. Wichtig ist auch zu wissen, dass GET Methodenaufrufe *safe* und *idempotent* sind. Das heisst, dass ein GET Request keine Seiteneffekte produzieren sollte und mehrfach angewandt werden kann.

Die Antwort auf einen GET Request besitzt auch einen fixen Aufbau:

```
HTTP/1.1 200 Document follows
  Date: Tue, 14 Apr 1997 09:25:19 PST
  Server: JWS/1.1
  Last-modified: Mon, 17 Jun 1996 21:53:08 GMT
  Content-type: text/html
  Content-length: 4435

  <4435 Bytes Daten -- der Dokument Body>
```

Die HEAD Methode

Die HTTP HEAD Methode ist ähnlich aufgebaut wie die GET Methode: der Request dieht genau gleich aus, einfach GET durch HEAD ersetzt.

Aber der Server liefert lediglich die Header Information zurück. Typische Einsatzmöglichkeiten für den HEAD Request sind:

- bestimmen des letzten Modifikationsdatums des Dokuments (Caching)
- die Grösse des Dokuments (vor dem Herunterladen, damit der Fortschritt angezeigt werden kann.
- der Server Typ, um die Client Anfragen zu optimieren.
- den Typus des verlangten Dokuments, um sicher zu sein, dass der Client diesen Dokumenttyp unterstützt.

Auch HEAD soll safe und idempotent sein.

Die POST Methode

Ein HTTP POST Request gestattet es einem Client, Daten an den Server zu senden:

- ablegen einer Message bei einer Newsgruppe
- eintragen von Infos in einem Gästebuch
- mehr Informationen transferieren, als ein GET Request gestatten würde

Der Unterschied zwischen GET und POST ist signifikant: ein GET Request kann nur wenig Informationen zum Server / vom Server verschieben. POST ist flexibler und kann auch einen Input Stream unfassen.

Hier ein typischer POST Request:

```
  POST /servlet/MeinServlet HTTP/1.1
  User-Agent: Mozilla/4.0 (
   compatible;
   MSIE 4.01;
   Windows NT)
  Host: www.joller-voss.ch
  Accept: image/gif, image/x-xbitmap,
   image/jpeg, image/pjpeg, */
  Content-type: application/x-www-form-urlencoded
  Content-length: 39

  name=Joller&company=Joller-Voss
```

Die Leerzeile signalisiert das Ende des POST Requests und den Beginn der erweiterten Information. POST ist weder safe noch idempotent, da die Daten modifiziert werden könnten und damit nicht mehr unverändert zur Verfügung stehen.

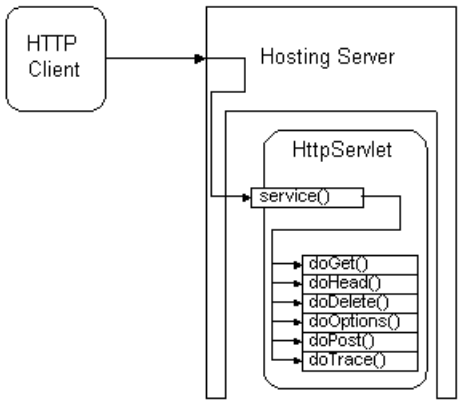
HTTP Support Klassen

Nachdem Sie nun in einem schnellen Überflug etwas Informationen zu den HTTP Methoden erhalten haben, wollen wir sehen, wie Java in den Servlet Packages diese Methoden und das HTTP Protokoll unterstützt.

Die abstrakte Klasse `javax.servlet.http.HttpServlet` stellt eine Implementation des `javax.servlet.Servlet` Interface zur Verfügung und umfasst viele weitere hilfreiche Funktionen / Methoden. Falls man einfache Servlets schreiben möchte, bietet diese Kasse eine gute Basis für eine eigene Implementation. Die Klasse `HttpServlet` stellt eine Implementation der Methode `service()` zur Verfügung. Die folgenden Methoden stehen in direkter Verbindung mit den entsprechenden HTTP Requests:

- `doGet()`
- `doHead()`
- `doDelete()`
- `doOptions()`
- `doPost()`
- `doTrace()`

Das folgende Diagramm zeigt, dass Sie service() Methode die verschiedenen HTTP Request interpretieren kann:



Die Klasse `HttpServlet` ist in Wirklichkeit recht clever. Sie stellt beispielsweise fest, welche Methoden in Unterklassen überschrieben wurden und kann einem Kunden Informationen über die Fähikeiten des Servers liefern.

Einsatz der HTTP Support Klassen

Falls Sie die HTTP Support Klassen einsetzen, kreieren Sie in der Regel ein neues Servlet, welches `HttpServlet` erweitert und eine der Methoden `doGet()` oder `doPost()` oder beide überschreibt. Auch andere Methoden können überschrieben werden, falls man eine detailliertere Kontrolle benötigt.

Die HTTP Methoden verwenden zwei Argumente:
1) ein `HttpServletRequest` Objekt und
2) ein `HttpServletResponse` Objekt.

Die `HttpServletRequest` Klasse besitzt verschiedene Methoden, welche sehr praktisch sind, beispielsweise um Requests zu parsen oder den Text aus einem Request zu lesene.

Die `doGet()` Methode sollte:

- die Daten aus dem Request lesen (Eingabe Parameter)
- die Response Header setzen (Länge, Typus, Kodierung)
- die Antwortdaten schreiben

Dass GET safe sein sollte hängt damit zusammen, dass GET Daten anschaut ohne sie zu verändern, wie dies beim POST Request geschieht.

Falls Sie ein Formular bearbeiten müssen, sollten Sie die `doPost()` Methode implementieren.

Die HEAD Requests werden mit der `doGet()` Methode der `HttpServlet` Klasse behandelt. Effizienter ist allerdings ein separates Behandeln der GET und HEAD Requests.

1.1.4.4. Zusammenfassung

Das Java Servlet API ist eine *Standard Extension*. Das heisst, dass es eine klare Definition der Servlet Interfaces gibt, welche aber nicht Teil des J2SE JDK der Standard Edition von Java sein müssen. Dies ist aber nicht definitiv, wie man am Beispiel von XML sehen kann (XML war eine Java Extension, wurde aber aus Marketing Gründen in die Standard Edition verschoben).

Neu ist, dass Sun die Servlet Technologie nicht mehr separat als JSDK oder JWS, sondern als Teil der Enterprise Edition J2EE ausgeliefert.

javax.servlet: Genereller Servlet Support

`Servlet`	Interface, welches die Kommunikation zwischen Server und Servlet definiert: init(), service() und destroy() Methoden..
`ServletConfig`	Interface, welches die Konfigurationsparameter für ein Servlet beschreibt. Diese werden an die init() Methode übergeben. Das Servlet sollte eine Referenz auf das ServletConfig Objekt speichern und eine getServletConfig() Methode definieren, welche diese Infos liefert, falls verlangt..
`ServletContext`	Interface, welches beschreibt, wie das Servlet Informationen über den Server erhalten kann. Die Methode getServletContext() des ServletConfig Objekts liefert die Infos.
`ServletRequest`	Interface, welches Informationen über Client Requests beschreibt.
`ServletResponse`	Interface, welches Informationen an den Client zurückliefert..
`GenericServlet`	Basis Servlet Implementaton, welche die ServletConfig Objekt Referenz, und und mehrere Methoden implementiert. init() und destroy().werden als leere Methoden zur Verfügung gestellt.
`ServletInputStream`	Unterklasse von InputStream zum Lesen der Daten aus einem Client Request. Umfasst auch eine readLine() Methode
`ServletOutputStream`	Ein OutputStream in den Antworten an den Client geschrieben werden..
`ServletException`	Wird bei Servlet Problemen geworfen
`UnavailableException`	Wird geworfen, falls ein Servelt nicht verfügbar ist.

javax.servlet.http: Support für HTTP Servlets

`HttpServletRequest`	Unterklasse von ServletRequest welche mehrere Methoden definiert, mit deren Hilfe die HTTP Request Headers geparsed werden können..
`HttpServletResponse`	Unterklasse von ServletResponse welche Zugriff und Interpretation der HTTP Status Codes und Header Information gestattet.
`HttpServlet`	Unterklasse von GenericServlet welche die HTTP Requests aufteilt:.ein HTTP GET Request wird an die service() Methode weitergeleitet und von der doGet() Methode bearbeitet.
`HttpUtils`	Klasse, welche beim Parsen der GET und POST Requests hilft.

1.1.5. Servlet Beispiele

Nun schauen wir uns einige typische Beispiele für den Einsatz von Servlets an:

- Generierung von Inline Content
- Verarbeitung von HTTP Post Requests
- Einsatz von Cookies
- Session Information
- Datenbankverbindung

1.1.5.1. Generierung von Inline Content

Oft enthält eine Web Seite lediglich kleine Informationsstücke, welche zur Laufzeit spezifiziert werden müssen. Das meiste an der Seite kann statisch bleiben. Daher bietet es sich an, lediglich jene aktuellen Daten zu rgänzen. Einige Web Server unterstützen sogenanntes Server Side Includes oder SSI. In der Regel verwenden die Server für Web Seiten, welche dafür verwendet werden, einen speziellen Dateityp, shtml. Damit weiss der Server, dass er auf spezielle Tags treffen kann. In JWS, dem Java Web Server, können Sie einen speziellen Tag, <servlet...> verwenden:

```
<servlet code="DatePrintServlet">
    <param name=timezone value=pst>
  </servlet>
```

Dieser Tag sendet dem Server die Nachricht, dass er ein Servlet, das DatePrintServlet, starten soll, um inline Content zu generieren.

SSI und Servlets gestatten es einem HTML Designer, ein Skeleton für eine Seite zu schreiben und passend zu ergänzen, also nicht die ganze Seite im Servlet zur Verfügung zu stellen. Beispiele für solche Einsätze wären Besucherzähler auf einer Seite. Auch das DatePrintServlet hat eine ähnliche Funktion: es muss einfach das aktuelle Datum auf die HTML Seite ausgeben. Der Ausgabe MIME Typ ist dabei "text/plain", nicht "text/html".

1.1.5.2. Übung - Inline Content

Arbeiten Sie die Übung zum Inline Content im Übungsteil durch.

1.1.5.3. Verarbeitung von HTTP Post Requests

HTTP POST Requests werden anders als GET Requests bearbeitet. Einer der Gründe ist sicher der, dass mit dem POST Request Daten auf dem Server verändert werden. Zudem kann der Informationsfluss vom Client zum Server beträchtlich sein. Daher muss die `doPost()` Methode einen Input Stream oder Reader öffnen, um alle Informationen vom Client zu lesen. Beim POST Request werden die Werte zu den Parametern nicht in der URL mitgeliefert.

Das Problem der Transaktionen wurde bei den DBMS bestens gelöst. HTTP funktioniert aber grundlegend anders. Transaktionen benötigen so etwas wie einen Zustand, und das liefert HTTP nicht. Die Lösung dieses Dilemmas ist nie einfach und in der Regel nicht sehr schön. Servlets bieten einen Mechanismus an, mit dem die Session Information erhalten werden kann. Wir kommen weiter unten nochmal auf diese Probleme zurück.

Falls diese Aufgabe gelöst ist, kann man sich an die Analyse des Input Streams machen. Dieser muss als Name / Wert Paar aus dem Formular gelesen werden. Die Hilfsklasse `HttpUtils` liefert die Methode `parsePostData()`, welche einen Input Stream als Parameter vom Client akzeptiert und einen Hashtable mit den Parametern liefert. Falls ein Parameter mehrwertig ist, liefert die Hashtable ein Array zum entsprechenden Namen.

1.1.5.4. Übung - HTML Formulare

Arbeiten Sie die Übung zum Thema HTML Formulare im Übungsteil durch.

1.1.5.5. Cookies

Falls Sie Cookies noch nicht kennen: ein Cookie ist ein benanntes Datenelement, welches vom Browser unterhalten wird, normalerweise für das Session Management. Da HTTP Verbindungen zustandslos sind, können Sie ein Cookie benutzen, um Daten über mehrere HTTP Verbindungen zu speichern. Die `Cookie` Klasse ist der Ort, wo alles geschieht. Die `HttpSession` Klasse beschreiben wir im nächsten Abschnitt. Sie ist einfacher einzusetzen, unterstützt aber kein Session Tracking über mehrere Browser Sessions.

Damit Sie Cookie Informationen speichern können, müssen Sie ein Cookie kreieren und initialisieren und beim Senden der Antwort einbeziehen. Das Cookie wird als Teil des Headers an den Client gesandt.

```
private static final String SUM_KEY = "sum";
...
int sum = ...; // ...
Cookie dasCookie = new Cookie (SUM_KEY, Integer.toString(sum));
response.setContentType("text/html");
response.addCookie(dasCookie);
```

Hier einige Spezialitäten der Cookies:

1) alle Informationen, die für ein Cookie bestimmt sind, müssen Textdaten sein. Daher müssen wir oben die Integer Zahl in eine Zeichenkette umwandeln.
2) ein Cookie lebt in der Regel nur für eine Browser Session. Falls dies nicht reicht, können Sie mit der Methode `setMaxAge(intervall)` ein längeres Leben verlangen.

3) falls intervall negativ ist, dann wird das Cookie zerstört, sobald der Brwoser verlassen wird. Falls der Wert 0 ist, wird das Cookie sofort wieder gelöscht.

Das Lesen von Cookies ist komplexer. Sie können das Cookie nicht mit einem speziellen Schlüssel lesen. Es kann auch sein, dass mehrere Cookies den selben Namen haben.

```
int sum = 0;
  Cookie dasCookie = null;
  Cookie cookies[] = request.getCookies();
  if (cookies != null) {
    for(int i=0, n=cookies.length; i < n; i++) {
      dasCookie = cookies[i];
      if (dasCookie.getName().equals(SUM_KEY)) {
        try {
          sum = Integer.parseInt(dasCookie.getValue());
        } catch (NumberFormatException ignored) {
          sum = 0;
        }
        break;
      }
    }
  }
```

Sie finden ein Testprogramm auf dem Server / der CD.

1.1.5.6. Session Information

Eine Session ist eine kontinuierliche Verbindung eines fixen Browsers über einen bestimmten Zeitraum. Jeder Web Server kann diese Zeit selber auch noch setzen. JWS setzt die Standard Session Zeit auf 30 Minuten. Mit Brower Cookies kann man die Session Informationen erhalten. HTTP Servlets steht dafür die Klase `HttpSession` zur Verfügung. Die `HttpServletRequest` Klasse liefert Informationen über die aktuelle Session mit Hilfe der Methode `getSession(boolean)`. Falls der Parameter `true` ist, wird eine neue Session kreiert.

```
public void doGet (HttpServletRequest request,
    HttpServletResponse response)
    throws ServletException, IOException {
  HttpSession session = request.getSession(true);
  // ...
```

Falls Sie Zugriff auf eine HttpSession haben, können Sie Session spezifischen Daten als Name / Wert Paare abspeichern. Auf den Zeitpunkt zu dem die Session kreiert wurde, können Sie mit der Methode `getCreationTime()` und auf die letzte Zugriffszeit mit `getLastAccessedTime()` zugreifen.

Um Daten zu speichern, kann man die Methode `putValue(key, value)` einsetzen. Falls Sie Informationen benötigen, steht Ihnen dafür die Methode `getValue(key)` zur Verfügung..

```
private static final String SUM_KEY =
  "session.sum";
private static final String ERROR_KEY =
  "session.errors";
Integer sum = (Integer) session.getValue(SUM_KEY);
int ival = 0;
if (sum != null) {
  ival = sum.intValue();
}
try {
  String addendString =
  request.getParameter("Addend");
  int addend = Integer.parseInt (addendString);
  sum = new Integer(ival + addend);
  session.putValue (SUM_KEY, sum);
} catch (NumberFormatException e) {
  Integer errorCount =
    (Integer)session.getValue(ERROR_KEY);
  if (errorCount == null) {
    errorCount = new Integer(1);
  } else {
    errorCount = new Integer(errorCount.intValue()+1);
  }
  session.putValue (ERROR_KEY, errorCount);
}
```

Wie mit allen Servlets: eines Tages müssen Sie eine Ausgabe produzieren. Sessionspezifisch geschieht dies wie im folgenden Programmfragment skizziert:

```
response.setContentType("text/html");
PrintWriter out = response.getWriter();
out.println("<html>" +
  "<head><title>Session Information</title></head>" +
  "<body bgcolor=\"#FFFFFF\">" +
  "<h1>Session Information</h1><table>");
out.println ("<tr><td>Identifier</td>");
out.println ("<td>" + session.getId() + "</td></tr>");
out.println ("<tr><td>Created</td>");
out.println ("<td>" + new Date(
  session.getCreationTime()) + "</td></tr>");
out.println ("<tr><td>Last Accessed</td>");
out.println ("<td>" + new Date(
  session.getLastAccessedTime()) + "</td></tr>");
out.println ("<tr><td>New Session?</td>");
out.println ("<td>" + session.isNew() + "</td></tr>");
String names[] = session.getValueNames();
for (int i=0, n=names.length; i<n; i++) {
  out.println ("<tr><td>" + names[i] + "</td>");
  out.println ("<td>" + session.getValue (names[i])
    + "</td></tr>");
}
out.println("</table></center></body></html>");
out.close();
```

Sie finden ein vollständiges Beispiel auf dem Server / der CD.

1.1.5.7. Verbindung zu Datenbanken

Eine der typischen Anwendungen für Servlets sind Datenbankverbindungen über JDBC. Damit können Sie Datenbankzugriffe besser kontrollieren. Zudem können Sie auch die Nutzung der Datenbanklizenzen optimieren und DBMS Verbindungen poolen.

Das folgende Beispiel zeigt den grundsätzlichen Aufbau einer solchen Lösung.

- Verbindungsaufbau zur Datenbank in der `init()` Methode

```
Connection con = null;
public void init (ServletConfig cfg)
  throws ServletException {
  super.init (cfg);
  // Load driver
  String name = cfg.getInitParameter("driver");
  Class.forName(name);
  // Get Connection
  con = DriverManager.getConnection (urlString);
}
```

- Lesen der Datenbank Informationen in der doGet() Methode

```
public void doGet (HttpServletRequest request,
    HttpServletResponse response)
    throws ServletException, IOException {
  response.setContentType("text/html");
  // Reload
```

```
    response.setHeader ("Expires",
      "Mon, 01 Jan 1990 00:00:00 GMT");

    Statement stmt = null;
    ResultSet result = null;

    try {
      // Submit Query
      stmt = con.createStatement();
      result = stmt.executeQuery (
        "SELECT programmer, cups " +
        "FROM CAFFEELIST ORDER BY cups DESC;");

      // Kreiere Output
      PrintWriter out = response.getWriter();
      while(result.next()) {
        // generiere Ausgaben aus dem ResultSet
      }
    } finally {
      if (result != null) {
        result.close();
      }
      if (stmt != null) {
        stmt.close();
      }
    }
    out.flush();
    out.close();
  }
```

- Datenbankverbindung in der `destroy()` Methode abbauen

```
public void destroy() {
  super.destroy();
  con.close();
}
```

Beachten Sie, dass es schlechte Praxis wäre, eine Datenbankverbindung dauernd geöffnet zu lassen. Daher sollten Sie solche Verbindungen nicht in permanente Servlets einbetten.

1.1.6. Security Fragen

Wie bei den Applets sollten Sie sich Gedanken zur Sicherheit machen, falls Sie Servlets einsetzen wollen.

1.1.6.1. Die Servlet Sandbox

Ein Servlet kann im Prinzip aus verschiedenen Quellen stammen:von einem Benutzer, von einem Webmaster oder von einer Firma oder direkt heruntergeladen von einem anderen Server.

Je nach Quelle des Servlets kann ein bestimmter Trustlevel festgelegt werden. Das Konzept ist analog zum Security Modell der Applets.

Eine Servlet "Sandbox" ist ein Bereich, in dem Servlets lediglich beschränkte Rechte haben. Man kann ihnen aber auch zusätzliche Rechte geben. Entscheidend ist der Webmaster, der Verwalter des Web Servers. Im schlimmsten Fall könnte ein Servlet den Befehl `System.exit()` absetzen und das System oder den Web Server herunterfahren. Daher ist ein sinnvolles Sicherheitsmodell angebracht. Schauen Sie sich die Möglichkeiten des JWS an. Aber auch Jakarta verfügt über eine Struktur, welche es einem Benutzer schwierig machen. Servlets einzuschleusen.

1.1.6.2. Access Control Lists (ACLs)

Viele Web Server, beispielsweise Microsoft, bieten die Möglichkeit sogenannte Access Control Lists ACLs für den Zugriff auf bestimmte Seiten oder Verzeichnisse zu definieren und die Zugriffe einzuschränken oder zu kontrollieren.

Eine ACL legt fest, wer was wann wo machen darf:

- welche Zugriffe sind erlaubt?
- auf welche Objekte sollen die Regeln angewandt werden
- welche Benutzer haben welche Rechte

Die Definition der ACL ist auch von Web Server zu Web Server unterschiedlich. Aber in der Regel gestatten Web Server die Registrierung von Benutzern, mit entsprechenden Rechten.

ACLs sind sehr wichtig, da Servlets sehr viel Zugriffsrechte auf sensitive Daten haben könnten.

1.1.7. Threading Fragen

Ein Web Server kann die `service()` Methode eines Servlets für mehrere Requests gleichzeitig aufrufen. Daher muss man sich Fragen stellen in Bezug auf Thread Sicherheit.

Über die `init()` Methode brauchen Sie sich keine Sorgen zu machen. Die `init()` Methode kann lediglich einmal aufgerufen werden. Und dies gilt dann für die Lebensdauer des Servlets.

Die Methoden `service()` und `destroy()` werden erst ans Servlet geschickt, sofern `init()` erfolgreich abgeschlossen wurde.

Schwieriger wird es mit der Methode `service()`. Diese Methode kann von einem Server für mehrere Client Anfragen aufgerufen werden. Falls Sie das Servlet als `SingleThreadModel` deklariert haben, wird Multithreading verunmöglicht.

Falls Ihre Servlets Ressourcen ausserhalb des Web Servers nutzen, müssen Sie, niemand anderes darauf achten, dass keine Zugriffskonflike entstehen können.

Hier ein Beispiel: Sie wollen einen Zähler auf Ihren Seiten installieren

```
private int counter = 0;
```

Nun programmieren Sie die `service()` Methode:

```
int meineNummer = counter + 1;  // Zeile 1
  counter = meineNummer;          // Zeile 2

  // Rest der service() Methode

  counter = counter - 1;
```

Nun können Sie sich sehr leicht überlegen, was passieren kann, wenn beide Aufrufe Zeile 1 und dann Zeile 2 ausführen. In diesem Fall wird die Nummer falsch sein.

Eine Möglichkeit besteht darin, den Zugriff zu synchronisieren:

```
synchronized(this) {
    meineNummer = counter + 1;
    counter = meineNummer;
  }

// Rest der service() Methode

  synchronized(this) {
    counter = counter - 1 ;
  }
```

Damit haben wir die Konsistenz der Daten sichergestellt. Aber die Lösung ist eher bieder. Falls Sie mehr zum Thema lernen wollen, sollten Sie beispielsweise das Buch von Doug Lea *Concurrent Programming in Java* ansehen. Das Buch ist nicht einfach zu lesen!

1.1.8. JSDK 1.0 und JSDK 2.0 und Jakarta

Wie sieht die Entwicklung der Servlet Klassen aus?

Eigentlich wurden im neusten Release von Tomcat / Jakarta mehrere Klassen hinzugefügt. Aber die alte Funktionalität wird weiter unterstützt.

Welche Version Sie konkret einsetzen können, hängt vom Server ab!

1.1.8.1. Servlet Features im JSDK 2.0

Der Übergang von JSDK 1 auf JSDK 2 brachte folgende Verbesserungen:

- das Interface `SingleThreadModel`, mit dem angegeben wird, dass die `service()` Methode nur im Single Thread Mode genutzt werden kann.
- `Reader` und `Writer` Zugriff auf `ServletRequest` und `ServletResponse`.
- HTTP Session Klassen
- Cookies Support
- mehr Funktionalität der Klasse `HttpServletResponse`
- Verlagerung / Delegation von DELETE, OPTIONS, PUT und TRACE in entsprechende Methoden in `HttpServlet`

Veraltete Methoden:

- getServlets() sollte ersetzt werden durch getServletNames()

1.1.8.2. Aktuelle Version 2.3

Wie bereits mehrfach erwähnt, wurde die JSP und die Servlet Technologie von Sun an Apache abgegeben. Das neuste Release, V2.3 (final draft) passt unter anderem die Notation den neueren Trends an. Beispielsweise ist die Rede von Servlet Containers, also Bereichen im Server, welche dem Servlet zur Verfügung stehen.

Servlet V 2.3 sind Teil der Java Enterprise Edition und müssen von jedem J2EE Server implementiert werden. Diese Integration in J2EE ist der wesentlich neue Teil in der neusten Servlet Spezifikation.

1.1.9. Zusatzinformationen

Hier einige Links auf Servlet Informationen:

- die Sun Support Seite:
  http://java.sun.com/products/servlet

- eine Liste der Server und 3rd Party Produkten zu Servlets
  http://java.sun.com/products/servlet/runners.html

- Informationen zum HTTP Protokoll
  http://www.w3c.org/Protocols

- Es wurden auch mehrere Bücher zum Thema veröffentlicht, von McGraw Hill, O'Reilly und Wrox, einige sind auch schon in deutsch erhältlich.

Eine Warnung:
einige der Servlet Bücher konzentrieren sich auf eigene Spezialprodukte, bei denen Servlets eine Rolle spielen, aber eben nicht das eigentliche Thema sind.