Kapitel 6

Servlet-Grundlagen

Nachdem wir in den vorangegangenen Kapitel die wichtigsten Basistechniken im Internet- und WWW-Umfeld vorgestellt haben, wollen wir nun in die konkrete Entwicklung verteilter Anwendungen einsteigen. Der zweite Teil dieses Buchs, der mit diesem Kapitel beginnt, beschäftigt sich mit dem Thema Servlets. Servlets sind eine Technologie, die speziell auf die Verwendung des World-Wide Web als Basistechnik ausgerichtet ist. Das Ziel dieses Kapitels ist es zunächst, uns mit den Grundlagen der Servlet-Technik vertraut zu machen, wobei wir davon ausgehen, dass die bereits vorgestellten Technologien wie WWW, Java, HTML und JDBC bekannt sind.

Wir beginnen unsere Ausführungen mit einer kurzen Einführung in die „geschichtliche“ (soweit man hier schon von Geschichte reden kann) Entwicklung der Servlet-Technologie. Anschließend zeigen wir, wie die Architektur verteilter Anwendungen, so wie sie bereits in Abschnitt 2.4 auf Seite 33 ausführlich beschrieben wurde, mit Servlets aussehen wird, wobei besonders auf die 3-Tier-Architektur abgehoben wird.

Um Servlets ausführen zu können, wird in den Web-Servern des Systems eine so genannte Servlet-Engine benötigt. Darauf geht der nächste Abschnitt ein, bevor es dann mit dem Thema Servlet-Objekte in Java zum ersten Mal richtig konkret wird. Spezieller auf die Verwendung von HTTP als Transportprotokoll sind die HTTP-Servlets als Unterklasse der generischen Servlet-Objekte zugeschnitten. Es wird zu sehen sein, dass diese Variante bereits eine recht komfortable Benutzer- bzw. Programmierschnittstelle anbietet. Als Abschluss des Grundlagenkapitels wird das Modell der Sessions eingeführt, das es Clients und Servern erlaubt, längerfristige Verbindungen auf der Basis von Servlets aufrecht zu erhalten.

6.1 Entwicklung und Aufgabe der Servlet-Technologie

Servlets haben eigentlich von ihrer Entwicklung zu einer wichtigen Technik bei der Erstellung verteilter Anwendungen her zwei Wurzeln, nämlich zum einen die Idee des *Common Gateway Interface (CGI)* und zum anderen die der *Applets*. Beide Techniken wurden schon kurz in Kapitel 2 auf Seite 9 besprochen. Sie stellen jeweils grundverschiedene

Ansätze dar, denn bei CGI handelt es sich um eine serverseitige Technik, während Applets kleine Anwendungsprogramme sind, die beim Client ablaufen. Beim CGI werden Programmaufrufe und Parameter über eine standardisierte Schnittstelle an den Web-Server gegeben, der für die Ausführung der jeweiligen Programme als separate Prozesse des Betriebssystems sorgt. Applets werden vom Web-Server in den Browser beim Client geladen und dann ausgeführt.

Servlets sind wie Java eine Entwicklung der Firma Sun Microsystems. Am Namen „Servlet“ lässt sich bereits die Verwandtschaft zum Applet ablesen. Servlets sind prinzipiell das Äquivalent zum Applet auf der Server-Seite. Sie sind ebenfalls in Java geschrieben, werden aber wie CGI-Skripte nach Aufruf durch einen Browser bzw. dessen Benutzer auf dem Server ausgeführt. Im Gegensatz zu CGI startet der Web-Server jedoch keinen eigenen Prozess; vielmehr wird das Servlet mit Hilfe einer in den Web-Server integrierten *Servlet-Engine* ausgeführt. Heute gibt es eine große Zahl von Web-Servern, die zur Ausführung von Servlets in der Lage sind; mehr dazu ist in Kapitel 7 auf Seite 123 zu finden.

Als weitere wichtige Eigenschaften der Technologie sind zu nennen:

- Sun stellt die Servlet-Technologie als Programmierschnittstelle (API) über eine Klassenbibliothek zur Verfügung.
- Über das API kann der Programmierer auf die Client-Anfrage sowie auf weitere Umgebungsvariablen zugreifen. Die Antwort wird in einen Datenstrom geschrieben und so an den Client zurückgeschickt.
- Das API unterstützt die so genannten *Cookies*, mit denen Server benutzerspezifische Informationen auf dem Client-Rechner ablegen und in der nächsten Sitzung wieder herunterladen kann.
- Das API untertützt *Sessions*, mit deren Hilfe eine Verbindung zwischen Client und Server eine einzelne HTTP-Anfrage überdauert. Zu Cookies und Sessions gibt Abschnitt 6.6 auf Seite 118 weitere Informationen.
- Das API legt nicht fest, ob die eigentliche Anwendung, auf die mittels Servlets meistens zugegriffen wird, im selben Prozess läuft wie der Web-Server oder vielleicht sogar auf einem anderen Rechner. Teil III des Buchs wird zeigen, wie sich hier in Kombination mit CORBA sehr mächtige Anwendungen zusammen bauen lassen.

Servlets dienen wie das CGI dazu, eine Schnittstelle zwischen dem World-Wide Web und existierenden (oder auch neu zu schreibenden) nicht web-basierten Anwendungen zu schaffen. Mit anderen Worten ist es das Ziel, Dienste, die in der verteilten Umgebung verfügbar sind, über eine einzige standardisierte und bekannte Schnittstelle, nämlich den Web-Browser, zur Verfügung zu stellen.

6.2 Architektur verteilter Anwendungen mit Servlets

In Kapitel 2 auf Seite 9 hatten wir uns verschiedene Architekturen verteilter Anwendungen angeschaut. Mit Hilfe von Servlets lassen sich auf elegante Weise 3-Tier-Architekturen implementieren. Abbildung 6.1 zeigt die Servlet-basierte Variante dieser Architektur.

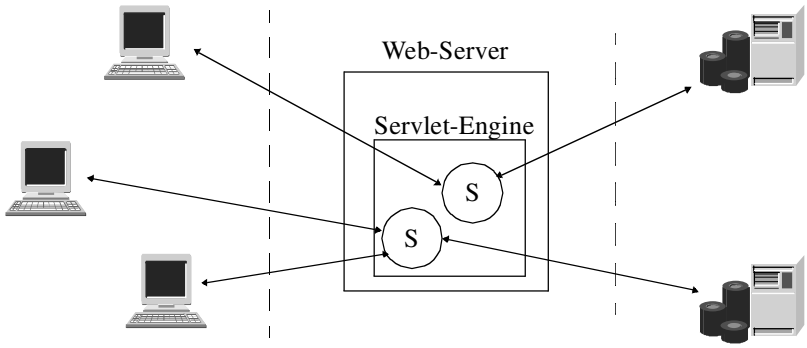

Abbildung 6.1: 3-Tier-Architekturen mit Servlets

In Tier 1 auf der Client-Seite laufen Web-Browser als Präsentationsprogramme. Durch Eingabe einer URL bzw. Klicken auf entsprechende Links werden Anfragen an den Web-Server übertragen. Dieser erkennt, dass durch die erhaltene URL ein Servlet-Aufruf kodiert wird. Der Aufruf wird an die Servlet-Engine übergeben, die wiederum für die Ausführung des angesprochenen Servlets sorgt. Zu den häufigsten Aufgaben des Servlets gehört es nun, die vom Client erhaltenen Parameter in die Sprache der eigentlichen Anwendung zu übersetzen und diese umkodierte Anfrage an die Anwendung in Tier 3 weiterzugeben. Diese bearbeitet die Anfrage, produziert gewöhnlich irgendein Ergebnis und gibt dieses über die im Servlet verwendeten Java-Methoden an das Servlet zurück. Dieses kodiert es wieder in die Sprache des Web-Browsers, also HTML, um und sendet es an den Client, der schließlich die Ergebnisse in seinem Ausgabefenster darstellt.

Eine der typischsten Anwendungsklassen, die mittels der in Abbildung 6.1 dargestellten Architektur implementiert wird, sind Datenbankanwendungen. Dem Benutzer steht dabei normalerweise ein Web-Formular zur Verfügung, über das er verschiedene Datenbankanfragen eingeben kann. Durch das Abschicken des Formulars (üblicherweise durch das Drücken einer Schaltfläche auf dem Formular) wird dessen Inhalt an das Servlet übertragen. Dieses dekodiert die Parameter, übersetzt sie in eine *SQL-Abfrage* und schickt die Anfrage mittels *JDBC* an die Datenbank. Die Datenbank führt die Abfrage durch, sammelt die Ergebnisse und gibt sie ebenfalls über JDBC an das Servlet zurück. Dem Servlet bleibt nur noch, die erhaltenen Datenbanktupel in HTML zu formatieren und als Ergebnisseite an den Browser zu schicken.

6.3 Servlet-Unterstützung in Web-Servern

6.3.1 Servlet-Engines

Die zentrale Komponente zur Ausführung von Servlets ist die Servlet-Engine. Um Servlets auf einem Web-Server installieren und ausführen zu könne, wird eine solche Engine unbedingt benötigt. Dabei handelt es sich prinzipiell um eine virtuelle Java-Maschine, die in der Lage ist, bestimmte Methoden von Objekten auszuführen, die als Servlet gekennzeichnet sind. Servlet-Engines sind sehr leicht einzubauen in Web-Server, die ohnehin schon in Java geschrieben sind, da hier die Servlets einfach mit dem Server zusammen als Java `class`-Dateien abgelegt werden können. Dazu gehört z.B. der Java Web-Server von Sun. Etwas aufwendiger gestaltet sich der Einbau in „traditionelle“ Web-Server wie Apache oder den Netscape Server, aber heute besitzen die meisten der verfügbaren Server auch eine Servlet-Engine.

6.3.2 Ablauf eines Servlet-Aufrufs

In der oben beschriebenen 3-Tier-Architektur gibt es einen relativ genau festgelegten Ablauf, wie Servlets ausgeführt werden bzw. wie sie andere Programme kontaktieren und Ergebnisse liefern. Dieser Ablauf ist grafisch in Abbildung 6.2 dargestellt.

Der Zugriff auf ein Servlet erfolgt gewöhnlich über ein Web-Formular, das in einem Browser-Fenster des Client dargestellt wird. Der Benutzer füllt dieses Formular aus und schickt es per Mausklick an den Web-Server. Dieser erkennt, dass es sich um den Aufruf eines Servlets handelt und ruft per Servlet-Engine das ausgewählte Servlet auf. Da es sich bei Servlets um nichts anderes als Java-Objekte handelt, muss natürlich eine bestimmte Methode des Objekts aufgerufen werden. Auf diese Details gehen wir im weiteren Verlauf dieses Kapitels detailliert ein.

Meistens wird die Aufgabe des Servlets darin bestehen, die per HTTP übertragenen Parameter in ein Format zu übersetzen, das für die Anwednung in Tier 3 verständlich ist. Anschließend werden die Informationen durch Aufruf einer API-Funktion an die Anwendung übertragen.

Sobald die Anwendung einen solchen Aufruf erhält, wird sie die gewünschte Programmfunktion ausführen und danach die Ergebnisse wiederum per API-Funktion an das Servlet zurückliefern. Das Servlet, das normalerweise in dieser Zeit auf die Ergebnisrückgabe durch die Anwendung wartet, kodiert die API-formatierten Ergebnisse in HTML um und schickt sie als Web-Dokument an den Client. Dieser stellt abschließend das erhaltene Dokument im Browser-Fenster dar.

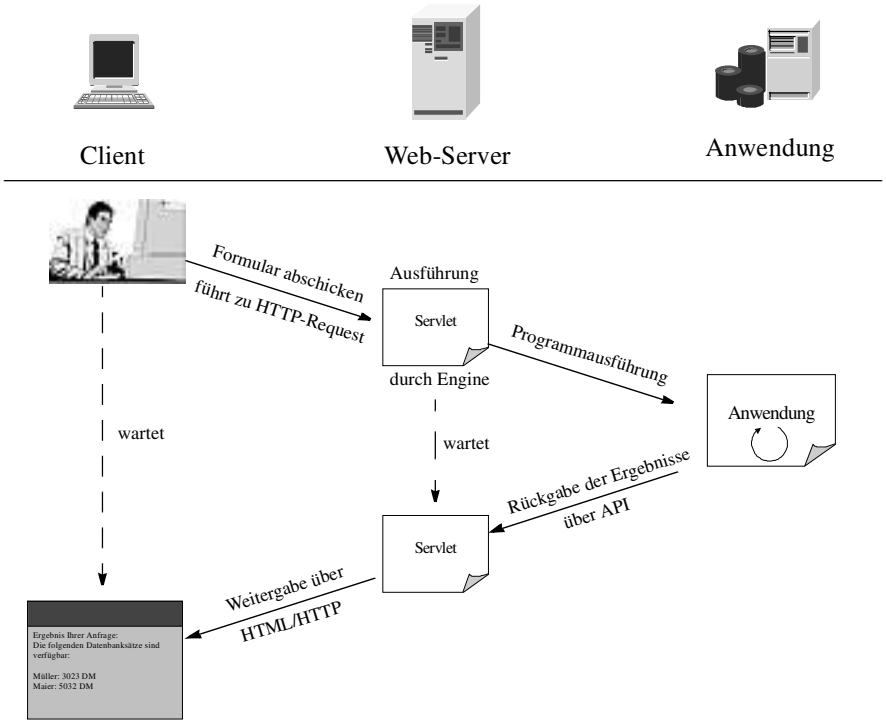

Abbildung 6.2: Ablauf eines Servlet-Aufruf

6.4 Generische Servlet-Objekte in Java

Wie sehen nun Servlet-Objekte aus, die die Arbeit in Tier 2 der Architektur verrichten? Dieser Abschnitt erläutert zunächst das generelle Aussehen der Servlet-Bibliothek in Java, um dann genauer auf den Lebenszyklus von Servlets einzugehen. Außerdem werden die speziellen Anfrage- und Antwort-Objekte betrachtet.

6.4.1 Die Servlet-Bibliothek in Java

Die Servlet-Bibliothek steht in Java im Paket `javax.servlet` zur Verfügung. Die zentrale Komponente ist das `interface Servlet`, das von allen Objekten, die als Servlet arbeiten sollen, implementiert werden muss. In diesem Kapitel werden nur die wichtigsten Komponenten der Klassen vorgestellt; eine vollständige Übersicht über alle Klassen und Methoden findet sich in Anhang B auf Seite 447.

Die wichtigste Methode dieser Schnittstelle heißt `service()`. Ihre Aufgabe liegt in der Entgegennahme, Verarbeitung und Beantwortung von Anfragen des Clients. Zu diesem Zweck besitzt sie folgendes Aussehen

```
void service(ServletRequest request, ServletResponse response);
```

Über die beiden Objekte vom Typ `ServletRequest` und `ServletResponse` bekommt das Servlet Zugriff auf Ein- und Ausgabedatenströme. Diese versorgen das Servlet mit Daten vom Server bzw. erlauben ihm, die Antwortdaten an den Client zurückzugeben.

Um dem Programmierer die Arbeit zu vereinfachen, wurden zuätzlich zwei abstrakte Klassen bereit gestellt, die die Schnittstelle implementieren und zusätzliche Funktionalität anbieten. Diese beiden Klassen heißen `GenericServlet` und `HttpServlet`. Erstere findet sich ebenfalls in `javax.servlet`, während die zweite unter `javax.servlet.http` abgelegt ist. Da die Klassen abstrakt deklariert sind, können sie nicht ohne weiteres benutzt werden. Vielmehr muss man sich eine eigene Klasse schreiben, die von einer dieser Klasse erbt und außerdem mindestens die Methode `service()` implementiert. Das `HttpServlet` wird noch genauer in Abschnitt 6.5 auf Seite 115 betrachtet. Das `GenericServlet` fügt dem `interface Servlet` vor allem Methoden zum Mitprotokollieren von Zugriffen hinzu.

Um also ein neues Servlet zu implementieren, muss eine Klasse erzeugt werden, die entweder

- selbst das `interface Servlet` implementiert, oder
- von der Klasse `GenericServlet` erbt, oder
- von der Klasse `HttpServlet` erbt.

Eine typische Klassendefinition sähe damit wie folgt aus:

```
class NeuesServlet extends GenericServlet {
   // service() muss implementiert werden
   public void service(ServletRequest req, ServletResponse res) {
      // lies req
      // führe Arbeiten durch
      // schreibe res
   }
```

Generell verwenden Anwendungen das `GenericServlet`; für Anwendungen, die auf der Verwendung von HTTP als Transportprotokoll und HTML als Seitenbeschreibungssprache basieren, bietet es sich an, die zusätzlichen Eigenschaften von `HttpServlet` zu nutzen. Betrachten wir aber zunächst noch weiter die generelle Funktionsweise von Servlets.

6.4.2 Lebenszyklus von Servlets

Ähnlich wie Applets durchlaufen auch Servlets einen bestimmten Lebenszyklus. An dessen Beginn werden sie einmal initialisiert. Dies geschieht durch den Aufruf der Methode `init()`, der von der Servlet-Engine durchgeführt wird, nachdem das Servlet erzeugt wurde. Anschließend steht das Servlet für Client-Anfragen zur Verfügung, die durch eine beliebige Anzahl von Aufrufen der Methode `service()` realisiert werden. Am Ende des Lebens einer Servlet-Instanz steht der Aufruf der Methode `destroy()`. Die drei Methoden sind alle bereits im `interface Servlet` definiert. `Service()` muss, wie schon gesagt, auf jeden Fall implementiert werden, während Standard-Implementierungen von `init()` und `destroy()` bereits durch die abstrakten Klassen bereit gestellt sind. Die Methoden müssen nur überschrieben werden, wenn andere oder zusätzliche Aktionen nötig sind.

6.4.3 Anfrageobjekte

Durch das `ServletRequest`-Objekt, das beim Aufruf der `service()`-Methode übergeben wird, erhält das Servlet Zugriff auf die eigentlichen Anfrageargumente des Clients, aber auch auf weitere Umgebungsvariablen. Dadurch entsteht insgesamt eine ähnliche Schnittstelle wie beim *Common Gateway Interface*.

Das Anfrageobjekt enthält als wichtigste Komponenten zwei Methoden, über die dem Servlet die Eingabeparameter des Clients zugänglich gemacht werden können. Diese Methoden haben die folgende Signatur:

```
Enumeration    getParameterNames();
String[]       getParameterValues(String name);
```

Die erste Methode liefert als Aufzählung alle verwendeten Parameternamen. Die zweite liefert zu jedem Namen ein Feld von Zeichenketten, denn jeder Parameter kann mehrere Werte haben. Falls die Parameternamen vorher bekannt sind und nur ein Parameterwert relevant ist, kann auch die Methode

```
String         getParameter(String name);
```

verwendet werden, die einfach einen Wert zu einem gegebenen Parameter liefert.

Damit lässt sich eine Standardmethode beschreiben, wie das Servlet die einzelnen Parameter und ihre Werte ausliest. Zunächst müssen die Parameternamen ermittelt werden:

```
public void service(ServletRequest req, ServletResponse res) {
   ...
   Enumeration e = req.getParameterNames();
```

Anschließend kann man durch die Parameter durchgehen und sich z.B. auf folgende Art und Weise alle Parameterwerte ausgeben lassen:

```
    while (e.hasMoreElements()) { //praktische Schleife für Enums!
       String name = (String)e.nextElement();
       String werte[] = req.getParameterValues(name);
       System.out.print("Name: " + name + " Werte:");
       if (werte != null)
          for (int i=0;i<werte.length;i++)
             System.out.print(werte[i] + " ");
       else
          System.out.print("(keine)");
       System.out.println();
    }
    ...
} // Ende von service()
```

Die entsprechenden Werte sind im Falle von Web-Formularen mit den Formulareingaben des Benutzers gleichzusetzen; im weiteren Ablauf der Methode `service()` müssen die Eingaben weiter verarbeitet werden. Kapitel 8 auf Seite 137 geht darauf noch ausführlich ein.

Auch Umgebungsvariablen können Servlets mit interessanten Informationen versorgen. Dazu besitzt `ServletRequest` eine Reihe weiterer Methoden, die dazu dienen, diese Variablen abzufragen. Beispiele dafür sind:

- `String getServerName()` zur Ermittlung des Host-Namen des Servers;
- `int getServerPort()` zur Ermittlung des TCP-Ports des Server-Prozesses;
- `String getRemoteAddr()` zur Ermittlung der IP-Adresse des Client-Rechners;
- `String getRemoteHost()` zur Ermittlung des Host-Namen des Client-Rechners.

Das Anfrage-Objekt besitzt noch weitere Methoden, die ebenfalls als Referenz detailliert in Anhang B auf Seite 447 aufgezählt werden.

6.4.4 Antwortobjekte

Das Gegenstück zum Anfrageobjekt ist das Antwortobjekt, das als Java-Klasse `ServletResponse` definiert ist. Seine Hauptaufgabe besteht darin, dem Servlet eine Möglichkeit zu geben, dem Client die Ergbnisse zu liefern. Dazu dient ein Ausgabedatenstrom, der über zwei Varianten angesprochen werden kann:

- Zum einen kann ein `ServletOutputStream` verwendet werden, der es erlaubt, binäre Daten zu senden. Hierzu bietet `ServletResponse` die Methode `getOutputStream()` an.

- Zum anderen kann ein `PrintWriter` angefordert werden, der die Übertragung von Text gestattet. Die Methode zum Anfordern dieses Objekts heisst `getWriter()`.

Bevor diese Methoden verwendet werden, sollte mittels `setContentType()` der MIME-Typ der Daten festgelegt werden. Vordefiniert ist „`text/plain`“. Gerade für Web-Anwendungen wird man oft eher „`text/html`“ verwenden wollen. In diesem Fall ist diese Methode aufzurufen, bevor der Ausgabedatenstrom angefordert wird.

Der folgende Code-Ausschnitt zeigt ein typisches Vorgehen bei der Festlegung der Ausgaben eines Servlets. Zunächst wird der MIME-Typ festgelegt und der Ausgabedatenstrom angefordert:

```
public void service() {
   ...
   res.setContentType("text/html");
   ServletOutputStream sout = res.getOutputStream();
```

Auf den Stream kann mit den üblichen Methoden wie `print()` und `println()` zugegriffen werden:

```
   sout.println("Testausgabe ...");
   sout.close();
```

Eine interessante Notiz am Rande: Sowohl `ServletRequest` wie auch `ServletResponse` sind eigentlich vom Typ `interface`. Deshalb müssen die beiden irgendwo implementiert werden, bevor sie benutzt werden können. Tatsächlich muss sich darum jedoch nicht der Servlet-Programmierer kümmern; vielmehr ist es die Aufgabe der Servlet-Engine, diese Implementierungen bereit zu stellen.

6.5 Spezielle Servlets für Web-Anwendungen

Generische Servlets sind schon relativ komfortabel zu benutzen, aber unter der Annahme einer eingeschränkteren Umgebung lassen sich noch diverse Verbesserungen erzielen. Servlets werden insbesondere im Umfeld des World-Wide Web eingesetzt, in dem HTTP als Transportprotokoll für Web-Dokumente verwendet wird, die in der Sprache HTML beschrieben sind. Für solche Umgebungen sind die speziellen Klassen und Schnittstellen des Pakets `javax.servlet.http` vorgesehen, die sich die Verfügbarkeit des HTTP-Protokolls zu Nutzen machen. Dieser Abschnitt beschreibt mit `HttpServlet`, `HttpServletRequest` und `HttpServletResponse` die drei grundlegenden Klassen.

6.5.1 HTTP-Servlets

Die Klasse `HttpServlet` ist eine Unterklasse von `GenericServlet` und erbt damit deren Methoden. Dazu definiert sie jedoch noch einige andere Methoden, die die Vorteile der Verwendung von HTTP ausnutzen.

HTTP ist, wie früher schon erläutert, ein textbasiertes Transportprotokoll für Hypertextdokumente. Die wichtigsten Protokollkommandos sind `GET` und `POST`, mit deren Hilfe ein Dokument angefordert bzw. Informationen zum Server geschickt werden können. Anstelle der `service()`-Methode bietet nun `HttpServlet` die Methoden `doGet()` und `doPost()` an, mit denen die jeweiligen Anfragetypen vom Client verarbeitet werden. Die `service()`-Methode gibt es immer noch, aber sie ist nur noch dafür zuständig, die eintreffenden Anfragen an die richtigen Methoden, also an `doGet()` oder `doPost()` (es gibt noch weitere, hier nicht relevante), zu verteilen. Sie sollte deshalb auch tunlichst nicht überschrieben werden.

Innerhalb der neuen Methoden passiert dann vom Grundsätzlichen her nichts anderes als früher in der `service()`-Methode: es werden die Eingaben vom Client gelesen und entsprechend verarbeitet. Möglicherweise werden weitere externe Programme aufgerufen und abschließend werden Ergebnisse an den Client zurückgeschickt.

Betrachten wir als Beispiel ein HTTP-Servlet `NeuesServlet`, das bereit ist zur Annahme von `GET`-Kommandos des HTTP-Protokolls:

```
class NeuesServlet extends HttpServlet {
   ...
   public void doGet(HttpServletRequest req,
                     HttpServletResponse res) {
      ...
      res.setContentType("text/html");
      PrintWriter sout = res.getWriter();
      sout.println("<H1>Ausgabe eines Servlets<H1>");
      ...
   }
}
```

Aus Ausgabe dieses Servlets würde der Client beim Aufruf mittels `GET` ein HTML-Dokument erhalten, das eine Überschrift der Ebene 1 im Browser-Fenster ausgibt.

In der Zeile, in der die Methode `doGet()` aufgeführt wird, ist zu sehen, dass auch die Anfrage- und Antwortobjekte spezielle HTTP-Versionen besitzen. Diese beiden werden in der Folge beschrieben.

6.5.2 HTTP-Anfrageobjekte

Das spezielle Objekt zur Behandlung HTTP-basierter Anfragen ist im Interface `HttpServletRequest` beschrieben. Die Verbesserungen gegeüber dem generischen `ServletRequest` liegen vor allem im erweiterten Zugriff auf Umgebungsvariablen und auf

typische HTTP-Komponenten wie den Header eines Dokuments. Ansonsten stehen die Methoden der Oberklasse zur Verfügung, die den Zugriff auf die vom Client gesendeten Feldnamen und -werte erlauben. Die Verwendung erfolgt dann auch auf die gleiche Art und Weise.

Zu den zusätzlichen Infos über die Umgebung gehören beispielsweise Informationen über die URL, in der der Servlet-Aufruf kodiert ist, oder der Name des Benutzers, soweit er sich authentifiziert hat. Eine interessante Methode, die in Abschnitt 6.6 auf Seite 118 noch eine Rolle spielen wird, stellt `getCookies()` dar, mit deren Hilfe sich längerfristige Beziehungen zwischen Client und Server herstellen lassen.

6.5.3 HTTP-Antwortobjekte

Etwas spannender ist das HTTP-spezifische Antwortobjekt `HttpServletResponse`, das die Methoden und Eigenschaften von `ServletResponse` erbt. Damit stehen dem Programmierer zunächst die gleichen Möglichkeiten der Produktion von Ausgaben an den Client zur Verfügung, insbesondere also eine `PrintWriter` oder ein `ServletOutputStream`.

Bei den neuen Komponenten von `HttpServletResponse` handelt es sich zum einen um eine größere Anzahl von Konstanten, die die HTTP-Status-Codes repräsentieren. Diese Status-Codes sieht der Benutzer von Web-Seiten vor allem dann, wenn eine Anfrage nicht funktioniert und auf dem Bildschirm eine kryptische Fehlermeldung der Art „HTTP 404 – File not Found“ erscheint. Es gibt eine ganze Reihe weiterer solcher Statusberichte, die den Benutzer über unterschiedliche Probleme unterrichten. Der günstigste Fall ist durch den Code „200“ beschrieben — in diesem Fall hat alles funktioniert.

Die entsprechenden Codes sind als Integer-Konstanten definiert und können mit den Funktionen

- `void sendError(int statuscode)` für kommentarlose Statusmeldungen oder
- `void sendError(int statuscode, String message)` für eine zusätzliche Nachricht

an den Client geschickt werden. Außerdem stehen eine Reihe weitere Methoden zur Verfügung, mit denen sich die HTTP-Header eines erzeugten Dokuments manipulieren lassen:

- `void setHeader(String name, String value)` setzt ein Header-Feld;
- `void setStatus(int statuscode)` setzt nur den Status eines Dokuments, ohne das Dokument schon abzusenden.

Als weitere wichtige Methode und als Gegenstück zur `getCookies()`-Methode des `HttpServletRequest`-Objekts bietet `HttpServletResponse` dem Programmie-

rer über `setCookies()` die Möglichkeit, Cookies auf dem Client-Rechner zu installieren. Mehr dazu ebenfalls im nächsten Abschnitt.

6.6 Längerfristige Verbindungen

Das Protokoll HTTP ist ein verbindungsloses Protokoll. Das bedeutet, dass ein Web-Server keinerlei Status-Informationen über einen Client speichert. Selbst wenn dieser wiederholt Seiten von diesem Server ab- oder Servlets aufruft, sehen diese Aufrufe für den Server jedesmal wie ein komplett neuer Aufruf durch einen neuen „Kunden“ aus. Nun kann man sich leicht vorstellen, dass es eine Menge Situationen gibt, in denen eine längerfristige Beziehung zwischen Client und Server sinnvoll ist, in denen der Server also Informationen über den Client speichert. Das typische Beispiel für eine solche Anwendung, das auch hier an späterer Stelle präsentiert wird, stellt die Einkaufswagen-Metapher dar. Ein Kunde eines elektronischen Ladens hat gewöhnlich die Möglichkeit, zunächst alle seine Einkäufe in einem solchen virtuellen Wagen abzulegen und am Ende seines Einkaufs alle gewählten Produkte auf einmal zu bezahlen. Wie aber soll ein derartiger Speicher implementiert werden, wenn sich der Server eigentlich keine der zu sammelnden Informationen zwischen zwei Auswahlentscheidungen merken kann? Denn jede Auswahl eines Produkts wird üblicherweise durch den Aufruf neuer Web-Seiten realisiert, also auf der Basis des Aufbaus einer komplett neuen HTTP-Kommunikation.

Zur Lösung dieses Problems wurden mit den so genannten *Cookies* sowie den *HTTP-Sessions* zwei Ansätze entwickelt, die im Rest dieses Kapitels vorgestellt werden.

6.6.1 Cookies

Unter Cookies versteht man kleine Informationsstücke, die der Server auf dem Rechner des Clients ablegt. Wenn der Client eine Web-Seite bzw. ein Servlet auf dem entsprechenden Server aufruft, dann werden die für diesen Kontext gespeicherten Cookies an den Server mit übertragen.

Die Methoden, die der Server verwenden kann, um auf Cookies zuzugreifen bzw. sie beim Client zu erzeugen, wurden oben schon kurz angesprochen. Mittels der Methode

```
void addCookie(Cookie c)
```

wird ein Cookie beim Client hinterlegt. Die Klasse `Cookie` repräsentiert eine Datenstruktur, die als zentrale Komponenten den Namen und den Wert des Cookies besitzt. Diese beiden Variablen müssen auch gesetzt werden, wenn ein Cookie erzeugt wird:

```
Cookie c = new Cookie("TestCookie.zaehler","0");
```

erzeugt einen Cookie, das den Namen „TestCookie.zaehler“ und den Wert „0“ hat. Um den Cookie in die Antwort an den Client zu integrieren, muss er über `HttpServletResponse` gesetzt werden:

```
public void doGet(...,HttpServletResponse res) {
   ...
   // hier wird der Cookie mittels new Cookie erzeugt
   // dann das Setzen:
   res.addCookie(c);
   ...
   // schließlich: Schreiben der Antwort mittels PrintWriter)
}
```

Das Auslesen von Cookie-Werten erfolgt dann auf folgende Weise mittels `getCookies()`, möglicherweise in einem anderen Servlet:

```
public void doGet(HttpServletRequest, ...) {
   ...
   Cookie[] cookies = req.getCookies();
   Cookie c;

   for (i=0;i<cookies.length,i++)
      if (cookies[i].getName().equals("TestCookie.zaehler")) {
         c = cookies[i];
         break; // unterbricht for-loop
      }
   ...
```

Der Wert des gefundenen Cookies kann dann mittels `getValue()` ermittelt werden:

```
   if (c != null) {
      String value = c.getValue();
```

und enstprechend weiterverarbeitet werden. Eine typische Aktion bestünde etwa darin, den Wert des Cookies zu verändern und an den Client zurückzuschicken.

Cookies haben nicht den besten Ruf, da viele Benutzer es ungern sehen, wenn beliebige Server Daten auf dem eigenen Rechner ablegen. Die Lebenszeit vieler Cookies ist auf die Lebenszeit des Browsers beim Client begrenzt. Der Server kann jedoch die Lebenszeit auch erhöhen, so dass die Cookies auf der Platte des Client abgelegt werden. Die Bedenken bestehen insbesondere deshalb, weil nicht wirklich kontrollierbar ist, welche Informationen der Server liefert. Eine Abhilfe schafft deshalb die zweite Lösung, die HTTP-Session.

6.6.2 Sessions

Das Verfahren, das hier angewendet wird, um eine Beziehung aufrechtzuerhalten, besteht in der Erzeugung eines Objekts, das von den Servlets verwaltet wird. In diesem Objekt

kann ein Servlet Informationen über die Beziehung zu einem speziellen Client ablegen. Ein solches Objekt vom Typ `HttpSession` überlebt den Aufruf eines Servlets, d.h., es kann zu späterer Zeit bei einem neuen Aufruf des Servlets wieder verwendet werden. Mit anderen Worten: Die wesentlichen Informationen über eine Client-Server-Beziehung liegen beim Server und nicht wie bei den Cookies beim Client.

Die Erzeugung eines neuen bzw. der Zugriff auf ein existierendes Session-Objekt erfolgt über die Methode

```
HttpSession getSession(boolean create)
```

die zur Schnittstelle `HttpServletRequest` gehört. Wenn bereits eine Session zu dem aufrufenden Client existiert, dann wird diese verwendet. Andernfalls wird der Parameter `create` überprüft. Ist er auf `true` gesetzt, wird eine neue Session erzeugt, ansonsten nicht.

Objekte vom Typ `HttpSession` besitzen einige Basismethoden und -variablen, die z.B. ihre Identifikationsnummer, ihr Erstellungsdatum oder das Datum des letzten Zugriffs festhalten bzw. ausgeben. Um jedoch beliebige Informationen über einen Client festhalten zu können, muss es auch eine Möglichkeit geben, Parameter und deren Werte in der Session abzulegen bzw. später wieder zu lesen. Dies geschieht über die folgenden Methoden:

- `void setAttribute(String name, Object value)` legt einen Parameter `name` in der Sitzung an und weist ihm den Wert `value` zu. Falls der Parameter schon existiert, wird dessen Wert durch den neuen Wert ersetzt.
- `Object getAttribute(String name)` liest den Wert des Parameters `name` und liefert ihn an den Aufrufer.
- `Enumeration getAttributeNames()` liefert die Namen aller in der Sitzung abgelegten Parameter und gibt sie als Aufzählung von Strings zurück.

Das folgende Beispiel zeigt kurz, wie die beschriebenen Methoden in der Praxis eingesetzt werden. Ausführlicher geht darauf Kapitel 8 auf Seite 137 ein. Der nun gezeigte Code erzeugt zuerst eine Session bzw. schaut, ob bereits eine Session vorhanden ist:

```
public class SessionBeispiel extends HttpServlet {
   ...
   public void doGet(HttpsServletRequest req,
                     HttpServletResponse res) {
      HttpSession session = req.getSession(true);
```

Die Aufgabe soll nun sein, sämtliche vom Client gelieferten Parameter und Werte (also etwa die Inhalte eines Formulars, das der Client ausgfüllt hat) in die Session zu übernehmen. Der Einfachheit halber wird immer nur ein Parameterwert übernommen, von dem wir außerdem annehmen, dass er vom Typ `String` ist. Es wird die Methode `setAttribute()` verwendet:

```
      // lies zunächst die vom Client geschickten Parameter
```

```
        Enumeration e1 = req.getParameterNames();
        // dann laufe durch diese Elemente und schreibe sie in die Session
        while (e1.hasMoreElements()) {
           String name = (String)e1.nextElement();
           String value = req.getParameter(name);
           session.setAttribute(name,value);
        }
```

Nun schicken wir alle in der Session gespeicherten Parameter an den Client zurück:

```
        PrintWriter pw = res.getWriter();
        pw.println("Sie haben folgende Parameter und Werte geschickt:");
        Enumeration e2 = session.getAttributeNames();
        while (e2.hasMoreElements()) {
           String name = (String)e2.nextElement();
           String value = (String)session.getAttribute(name);
           pw.println("Parameter: " + name + "Wert: " + value);
        }
        ...
     } // doGet
  } // SessionBeispiel
```

Wir hatten gesagt, dass eine Session den Aufruf eines Servlets überlebt. Wie aber kann man eine Session beenden, denn ein Servlet wird nicht sämtliche Sessions für immer offen halten wollen? Zu diesem Zweck ist mit einer Session immer auch eine maximale Periode der Inaktivität verbunden. Diese Zeit kann von der Session mit der Methode `setMaxInactiveInterval()` bestimmt werden. Wird die damit festgelegte Zeit überschritten, so wird die Session inaktiviert und identifiziert den Client in der Folge nicht mehr. Diese Maßnahme ist wichtig, um die Servlets einigermaßen effizient zu halten.

Es bleibt die Frage, wie die Session-Identifikationsnummer vom Client an den Server übertragen wird. Tatsächlich geschieht das über den Weg der URL, genauer gesagt über die Technik des *URL-Rewriting*. Dabei werden alle URLs, die ein Server innerhalb einer dynamisch generierten Seite an den Client schickt, modifiziert, indem ein zusätzlicher Parameter zur Identifizierung der Session angehängt wird. Damit wird z.B. aus einer URL

```
http://192.168.0.1:8080/servlet/SessionBeispiel
```

die den Aufruf eines Servlets innerhalb eines Formulars beinhaltet, die modifizierte URL

```
http://192.168.0.1:8080/servlet/
        SessionBeispiel?jrunsessionid=3248654786450974
```

Klickt nun der Client auf diese modifizierte URL, dann wird der Parameter `jrunsessionid` automatisch an den Server zurück übertragen, der damit den entsprechenden Client identifizieren und ihm wieder seine Session zuordnen kann. Für den Autor einer solchen Anwendung bedeutet die Verwendung von URL-Rewriting erhöhte Aufmerksamkeit, denn nur eine einzige vergessene URL, die nicht umgeschrieben wurde, bedeutet den

Zusammenbruch der Anwendung, da die Client-Session dann nicht mehr gefunden werden kann.

Eine zweite Variante besteht in der Speicherung der Session-ID beim Client, und zwar natürlich wieder als Cookie. Auf diese Weise lassen sich die beiden Techniken auf elegante Weise kombinieren, ohne dass der Client zu viele Informationen als Cookie speichern muss. Unser Beispiel in Kapitel 8 auf Seite 137 wird noch detaillierter zeigen, wie diese Kombination effizient eingesetzt werden kann.

Nach der Einführung in die Servlet-Technologie geht das nächste Kapitel nun auf die Unterstützung des Entwicklers von Servlet-basierten Anwendungen durch Werkzeuge ein.