

Servlet Debugging

Markus Völter, voelter@acm.org, www.voelter.de

Bei der Arbeit mit Servlets kommt man recht schnell an den Punkt, an dem man Servlets vernünftig testen oder debuggen will. Mit Hilfe des hier erläuterten Verfahrens lassen sich Servlets als normale Java-Anwendungen verwenden und in der gewohnten IDE debuggen.

Servlets sind in erster Näherung ganz normale Java-Klassen. Um jedoch im Servlet-fähigen Web-Server verwendet werden zu können, müssen sie von der Klasse `javax.servlet.http.HttpServlet` erben [1,2]. Für die Verwendung von Servlets sind drei Methoden der Klasse `HttpServlet` maßgebend, müssen also vom Benutzer überschrieben werden, um das gewünschte Verhalten des Servlets zu erzeugen.

- `void init(ServletConfig config)` wird vom Host-Server beim Laden des Servlets aufgerufen. Die Methode wird verwendet, um Ressourcen wie Dateien oder Datenbankverbindungen zu allozieren.
- `void destroy()` dient dazu, Aufräumarbeiten zu erledigen und wird vom Server vor Beendigung des Servers aufgerufen.
- `void service(HttpServletRequest req, HttpServletResponse resp)` ist die wichtigste Methode. Sie wird bei jedem Aufruf des Servlets (vom Client per URL aus) aufgerufen, und erledigt die eigentliche Arbeit. In dieser Methode bringt der Entwickler die eigentliche Funktionalität des Servlets unter.

Servlets werden wie oben erwähnt innerhalb eines Web-Servers ausgeführt, der die entsprechenden Methoden aufruft. Um Servlets testen und debuggen zu können, ist es allerdings von Vorteil, wenn die Dinge außerhalb der Server-Umgebung laufen, das heißt als Java-Anwendung. Man muß also erstens die Server-Umgebung simulieren und zweitens die Ausgaben und Parameterübergabemechanismen zwischen Server und Servlet nachahmen.

Die Methode `init` wird beim Laden des Servlets aufgerufen. Der Entwickler muß bei seiner Implementation von `init` unbedingt darauf achten, daß er `super.init(config)` aufruft, sodaß das Servlet im Server richtig geladen und initialisiert wird. Jedoch geht dieser Aufruf, wenn das Servlet als Anwendung läuft schief, da eben die Server-Umgebung fehlt. Daher ist der erste Schritt das einführen einer Variable `runsAsApplication`, die auf `true` gesetzt wird, wenn das Servlet als Anwendung läuft. Die Methode `init` kann dann folgendermaßen aussehen.

```
public void init(ServletConfig c ) {  
    if ( !runsAsApplication)  
        super.init(c);  
    /* Eigene Initialisierungen */  
}
```

```
}
```

Die Methode `destroy` bedarf keiner speziellen Behandlung, es muß nur auch hier darauf geachtet werden, daß keine Methoden von `super` aufgerufen werden. Dafür kann auch die Variable `runsAsApplication` verwendet werden.

Interessant wird die Sache bei der Methode `service`. Diese hat als Formalparameter ein Objekt des Typs `HttpServletRequest` und eines des Typs `HttpServletResponse`.

`HttpServletRequest` enthält die Informationen, die mit der Anfrage zusammenhängen, wie z.B. die Anforderungsmethode (`GET` oder `PUT`) oder das Protokoll. Am wichtigsten sind die Parameter, die dem Servlet per URL übergeben werden. Diese lassen sich in der Methode `service` mit `getParameter(String paramName)` abfragen.

Zur Ausgabe der Ergebnisse an den WWW-Server (also der generierten Seite) kann man sich vom `HttpServletResponse`-Objekt mit `getOutputStream()` einen Stream besorgen, mit Hilfe dessen man ausgeben kann. Im Falle des Servlets als `Application` sollen die Ausgaben auf der Standardausgabe erscheinen.

`HttpServletResponse` und `HttpServletRequest` sind Interfaces, man kann sich nun eben eigene Klassen implementieren um das gewünschte Verhalten zu erzeugen.

Die Klasse `AppHttpServletRequest` ist also die Implementierung des Interface `HttpServletRequest`, welche einen Request des Servers simuliert.

Eine Klasse, die ein Interface implementiert, muß alle Methoden des Interfaces implementieren. In vielen Fällen wird jedoch vom `HttpServletRequest` eigentlich nur der Parameterübergabemechanismus benutzt. Alle Methoden, die nicht benutzt werden sollen können so implementiert werden, daß sie eine Fehlermeldung erzeugen.

Für den Parameterübergabemechanismus kann man der Klasse `AppHttpServletRequest` eine `Hashtable` von Stringpaaren mitgeben. Die Methode `getParameter` kann also folgendermaßen implementiert werden:

```
publicString getParameter( String name) {  
    if ( params.containsKey( name )  
        return((String)params.get(name));  
}
```

Um nachher vom Hauptprogramm aus die Parameter zu setzen wird eine Methode benötigt, die `Hashtable` zu füllen:

```
public void addParam( String key, String value ) {  
    params.put(key,value);  
}
```

Für die Ausgabe des Ergebnisses muß das Interface `HttpServletResponse` implementiert werden. Auch hier gilt: Nur die verwendeten Methoden müssen mit sinnvollem Inhalt gefüllt werden, die anderen können einen Fehler erzeugen oder, wie z.B. `setContentType`, einfach nichts tun.

Am wichtigsten dürfte hier die Methode `getOutputStream` sein, die einen Stream zur „Ausgabe auf die Web-Seite“ zurückliefert. Dieser Stream ist vom Typ `ServletOutputStream`.

Von dieser Klasse wird zu Debug-Zwecken eine Klasse `AppServletOutputStream` abgeleitet, die alle Methoden der Klasse `ServletOutputStream` überschreibt, und zwar so, daß alle Ausgaben auf der Standardausgabe landen.

Einige Beispiele:

```
public void print(String s) throws IOException {
    System.out.print( s );
}

public void print(boolean b) throws IOException {
    System.out.print( b );
}

public void print(char c) throws IOException {
    System.out.print( c );
}
```

Ein Beispiel soll die Anwendung der oben eingeführten Klassen verdeutlichen:

```
package vs.util.servlet;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;

public class Telefonauskunft extends HttpServlet {

    public boolean runsAsApplication;

    public void init(ServletConfig config)
        throws ServletException {
        if (!asApplication) super.init(config);
        /* Datenbankverbindung aufbauen */
    }

    public void service( HttpServletRequest req,
        HttpServletResponse resp)
        throws ServletException, IOException {
        // Ausgabeformat setzen
        resp.setContentType("text/html");
        out = new PrintWriter( resp.getOutputStream() );
        // denen gesucht werden soll
        String name = req.getParameter("name");
    }
}
```

```
String telefonNummer = ``;
/* Datenbank abfragen, und telefonNummer entspr.
   auffüllen */
out.println(„<HTML><BODY>“);
out.println(„Die Nummer für „+name+“ ist „ +
            telefonNummer );
out.println(„</BODY></HTML>“);
}

public void destroy() {
    /* DB-Verbindung abbauen */
}

public static void main( String args[] ) {
    try {
        AppHttpServletRequest req =
            new AppHttpServletRequest();
        req.addParam( "name", "Hans Muster" );
        AppHttpServletResponse resp =
            new AppHttpServletResponse();
        SuchServlet w = new SuchServlet();
        w.runsAsApplication = true;
        w.init( null );
        w.service( req, resp );
        w.destroy();
    } catch ( Exception ex ) {
        ex.printStackTrace();
    }
}
}
```

Wird das Servlet vom Server aus geladen und ausgeführt, so hat die Funktion main() keine Aufgabe. RunAsApplication ist false, super.init() wird also aufgerufen. Die Parameter HttpServletRequest und HttpServletResponse sind die vom Server gelieferten Implementationen.

Wird das Servlet allerdings als Anwendung aufgerufen, so kommt die Methode main() ins Spiel. Diese erzeugt zunächst einen AppHttpServletRequest, dem dann per addParam ein Servlet-Parameter mitgegeben wird. Dann wird ein AppHttpServletResponse-Objekt erzeugt und runsAsApplication auf true gesetzt.

Dann wird der Web-Server simuliert: Die Methode init wird aufgerufen (mit null als Config, diese wird ja sowieso nicht weiterverwendet). Dann wird service mit den beiden soeben erzeugten Parametern aufgerufen, danach destroy().

Da dies nun eine ganz normale Anwendung darstellt, kann sie problemlos debuggt und getestet werden.

Literatur:

- [1] Sun Microsystems: Java Web Server <http://java.sun.com/products/java-server>