03-2014 | Herbst | www.ijug.eu

Java aktuell

Praxis. Wissen. Networking. Das Magazin für Entwickler

Aus der Community – für die Community

Java aktuell

Java im Mittelpunkt

Aktuell
NoSQL für Java

Performance
Java-Tuning

Pimp my Jenkins

Logging
Apache Log4j 2.0

Sonderdruck

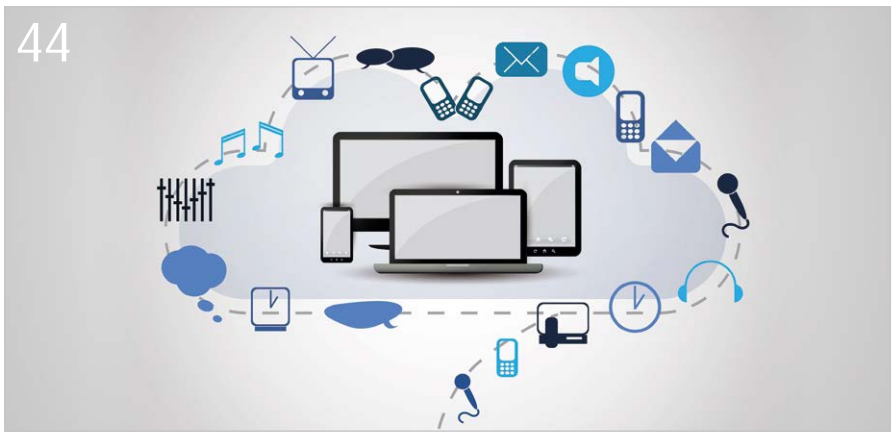
D: 4,90 EUR A: 5,60 EUR CH: 9,80 CHF Benelux: 5,80 EUR ISSN 2191-6977

4 191978 304903 02

iJUG Verbund

JavaLandConf 2014 Twitterwall

JavaFX– das neue Gesicht für Java-Anwendungen



Connectivity-as-a-Service für Cloud-basierte Datenquellen

Unbekannte Kostbarkeiten des SDK

Unbekannte Kostbarkeiten des SDK

Heute: HTTP-Server

Bernd Müller, Ostfalia

Das Java SDK enthält eine Reihe von Features, die wenig bekannt sind. Wären sie bekannt und würden sie verwendet, könnten Entwickler viel Arbeit und manchmal sogar zusätzliche Frameworks einsparen. Wir stellen in dieser Reihe derartige Features des SDK vor: die unbekannten Kostbarkeiten.

Das SDK enthält einen einfachen HTTP-Server, der ausreicht, um einfache Web-Seiten ausliefern zu können. Der HTTP-Server wird durch Möglichkeiten für sicheres HTTP, Benutzer-Authentifizierung und Filter erweitert.

Der HTTP-Server

Der HTTP-Server ist nicht im öffentlichen und damit offiziellen API des SDK enthalten, sondern im Package „com.sun.net.httpserver". Bei seiner Verwendung muss man sich über die Konsequenzen im Klaren sein: Es besteht keine Garantie dafür, dass der Server auch im nächsten Release enthalten und die Anwendung über die verschiedenen SDKs hinweg portabel ist.

Zumindest das zweite Manko kann man etwas entkräften: Sowohl das OpenJDK als auch die von Oracle distribuierte Erweiterung des OpenJDK enthalten den HTTP-Server. Oracles JRockit sowie IBMs JDK umfassen ihn ebenfalls.

Das Server-Package

Da das Package kein offizielles Java-API darstellt, ist das API-Doc nicht in der Distribution des Standard-SDK enthalten. Für eigene Versuche mit dem Package ist auf [1] verwiesen. Wir zitieren die API-Dokumentation des Package: „Provides a simple high-level Http server API, which can be used to build embedded HTTP servers."

Das Package enthält als zentrale Bestandteile die Klassen „HttpServer" und „HttpExchange" sowie das Interface „HttpHandler", die wir in unserem Beispiel gleich verwenden werden. Daneben sind noch weitere Klassen wie „Authenticator", „HttpPrincipal", „HttpsServer" und „Filter" enthalten, deren Namen den Leser auf weitere Realisierungsmöglichkeiten aufmerksam machen. Das Package enthält insgesamt ein Interface und siebzehn Klassen.

Die genannte Gefahr, dass das Package in zukünftigen Releases nicht enthalten ist, ist zumindest für den Package-Namen sehr real. Der Autor geht davon aus, dass alle „com.sun"-Artefakte über kurz oder lang auf OpenJDK-, Oracle- oder JCP-Bezeichner migriert werden, wie dies bereits in den XML-Namensräumen von Java-EE kürzlich erfolgt ist [2].

Eine einfache Server-Anwendung

Listing 1 zeigt eine einfache Server-Anwendung, die beispielsweise für eine Java-Client-Anwendung das Handbuch der Anwendung in Form von Web-Seiten publizieren kann. Die HTTP-Server-Anwendung wird durch eine Implementierung des Interface „HttpHandler" realisiert. Dieses Interface enthält die einzige Methode „handle()", die als Callback-Methode für eingehende HTTP-Requests fungiert. Die Klasse „SimpleHandler" implementiert „HttpHandler" und realisiert damit unseren einfachen HTTP-Server.

Die Methode „handle()" bekommt als einzigen Parameter einen „HttpExchange" übergeben. Diese Klasse kapselt empfangene HTTP-Requests und die zu erzeugende HTTP-Response. Sie enthält daher Methoden, um auf Details des Request und der Response zuzugreifen. Im Beispiel wird die HTTP-Methode abgefragt und nur bei „GET" eine Antwort generiert. Ebenfalls wird der URI des Request abgefragt und über einen einfachen Mechanismus eins zu eins auf ein Verzeichnis des Dateisystems abgebildet. Weitere Methoden erlauben beispielsweise den Zugriff

```
class SimpleHandler implements HttpHandler {
    private static final String ROOT = "...";
    public void handle(HttpExchange exchange) throws IOException {
        Headers responseHeaders = exchange.getResponseHeaders();
        String requestMethod = exchange.getRequestMethod();
        if (requestMethod.equalsIgnoreCase("GET")) {
            File file = new File(new File(ROOT), exchange.getRequestURI().toString());
            try (InputStream is = new FileInputStream(file);
                OutputStream responseBody = exchange.getResponseBody(); ) {
                responseHeaders.set("Content-Type", "text/html");
                exchange.sendResponseHeaders(HttpURLConnection.HTTP_OK, file.length());
                byte[] buffer = new byte[1024];
                int len;
                while ((len = is.read(buffer)) != -1) {
                    responseBody.write(buffer, 0, len);
                }
            } catch (FileNotFoundException e) {
                exchange.sendResponseHeaders(HttpURLConnection.HTTP_NOT_FOUND, 0);
            } catch (IOException e) {
                exchange.sendResponseHeaders(HttpURLConnection.HTTP_INTERNAL_ERROR, 0);
            }
        }
    }
}
```

Listing 1

auf das Principal-Objekt eines authentifizierten Benutzers, den Request-Header oder auf Client- und Server-IP-Adressen.

Das Beispielprogramm realisiert sogar eine einfache Fehlerbehandlung und benötigt trotzdem nur wenige Zeilen. Um das Beispiel zu vervollständigen, fehlt noch die Instanziierung der „HttpServer"-Klasse und die Bindung des oben definierten Handlers für eingehende Requests. Dies erfolgt in den Zeilen, die in der Main-Methode aufgerufen werden können (siehe Listing 2). Das Beispiel ist auf den vier genannten SDKs lauffähig, wobei uns JRockit nur für Java 6 vorliegt, sodass die Try-With-Resources-Anweisung umformuliert wurde.

Das Kompilieren mit „javac" gelingt ohne Probleme. Die Kompilate des Package sind in „rt.jar" vorhanden, sodass „javac" sie ohne weitere Konfiguration findet. Bei der Verwendung von Eclipse ist das Package zunächst nicht im Klassenpfad und man muss über „Window" –> „Preferences, Java" –> „Compiler" –> „Errors/Warnings", unter „Deprecated and Restricted API" den Eintrag „Forbidden Reference" auf „Ignore" oder „Warning" stellen.

```
InetSocketAddress addr = new InetSocketAddress(8080);
HttpServer server = HttpServer.create(addr, 0);
server.createContext("/", new SimpleHandler());
server.setExecutor(Executors.newCachedThreadPool());
server.start();
```

Listing 2

Fazit

Das SDK enthält einen einfachen HTTP-Server, der mit wenig Aufwand zu einem funktionsfähigen Web-Server ausgebaut werden kann. Obwohl der Server nicht im öffentlichen API enthalten ist und das Package mit dem Präfix „com.sun" beginnt, ist es in allen dem Autor zugänglichen SDKs (OpenJDK, Oracle, IBM, JRockit) enthalten.

Literatur

[1] http://docs.oracle.com/javase/6/docs/jre/api/net/httpserver/spec/com/sun/net/httpserver/package-summary.html.

[2] https://blogs.oracle.com/theaquarium/entry/java_ee_schema_namespace_moved.

Bernd Müller
bernd.mueller@ostfalia.de

Bernd Müller ist Professor für Software-Technik an der Ostfalia. Er ist Autor des Buches „JavaServer Faces 2.0" und Mitglied in der Expertengruppe des JSR 344 (JSF 2.2).