



Das iJUG Magazin

Java aktuell

Herbst 2011

Java aktuell
Magazin der Java-Community

Java überall

- Neu: Java SE7
- Interview mit Patrick Curran, Vorsitzender des JCP
- Für Android entwickeln
- Testen mit Arquillian
- Suchen mit Apache Solr



iJUG
Verbund

Erfahrungen, Ideen und Lösungen für Java-Entwickler

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

Sonderdruck

04/2011



- 3 Editorial
- 5 Die lange Reise von Java 7
Markus Eisele, msg systems ag
- 8 „Unbedingt die Specs lesen und Feedback geben ...“
Interview mit Patrick Curran, Vorsitzender des JCP
- 11 Das Java-Tagebuch
Andreas Badelt, DOAG Deutsche ORACLE-Anwendergruppe e.V.
- 15 Java überall
Oliver Szymanski, Source-Knights.com, stellv. Vorstandsvorsitzender des iJUG
- 17 „Java muss sich neuen Einsatz-Szenarien wie Cloud und Mobile Computing stellen ...“
Interview mit Dr. Mark Little, Red Hat
- 19 Arquillian
Frederik Mortensen
- 22 Suchen mit Apache Solr
Peter Karich, Pannous GmbH
- 26 „Ich denke, die Java-Community ist wie eine große Familie ...“
Interview mit Michael Hüttermann, Java User Group Köln
- 28 Android – Java macht mobil
Andreas Flügge, object systems GmbH
- 31 Hibernate im Projekteinsatz
Dirk Mahler, buschmais GbR
- 34 Semantisch-orientierte Programmierung mit Java
Oliver Böhm
- 37 Slice – Unterstützung für verteilte, partitionierte und heterogene Datenbanken mit OpenJPA
Bernd Müller, Ostfalia Hochschule für angewandte Wissenschaften, sowie Harald Wehr, MAN Truck & Bus AG
- 40 NetBeans Platform 7
gelesen von Jürgen Thierack
- 41 XPages – Ein neues Framework zur Entwicklung von Web-Anwendungen
Dr. Rolf Kremer, PAVONE AG
- 45 „Bereits jetzt zählen wir zu den führenden Java-Magazinen im deutschsprachigen Raum ...“
Interview mit Fried Saacke, Vorstandsvorsitzender des iJUG
- 46 Leichtgewichtige Authentifizierung mit OpenID
Sebastian Glandien, Acando GmbH
- 51 Java-Problem-Determination mit der IBM Support Assistant Workbench
Marc Bauer, IBM Deutschland GmbH
- 54 Java EE 7 – eine Reise in die Wolken
Peter Doschkinow, ORACLE Deutschland B.V. & Co. KG
- 57 Varianten-Entwicklung in 3D mit Object Teams
Dr. Stephan Herrmann, GK Software AG
- 60 Unbekannte Kostbarkeiten des SDK Heute: Der Service-Loader
Bernd Müller, Ostfalia
- 62 Rich Client Frontends für umfangreiche Unternehmensanwendungen
Björn Müller, CaptainCasa
- 61 Inserenten
- 53 Impressum



Kleiner Exkurs darüber, wo wir Java direkt oder indirekt überall antreffen, Seite 15

Dies ist ein Sonderdruck aus der Java aktuell. Er enthält einen ausgewählten Artikel aus der Ausgabe 04/2011. Das Veröffentlichen des PDFs bzw. die Verteilung eines Ausdrucks davon ist lizenzfrei erlaubt. Weitere Informationen unter www.ijug.eu



Leichtgewichtige Authentifizierung mit OpenID

Sebastian Glandien, Acando GmbH

OpenID ist ein Standard zur dezentralen Anmeldung an Web-Anwendungen. Wurde bisher die Überprüfung einer Anmeldung immer durch die eigene Anwendung vorgenommen, delegiert OpenID dies an einen vertrauenswürdigen Dritten. Somit kann sich die Nutzerbasis eines Partners schnell in Synergie-Effekten auszahlen. OpenID schreibt die Ausstellung einer beglaubigten Nutzeranmeldung, deren Überprüfung und die Abfrage von zusätzlichen Nutzerattributen fest. Das offene Protokoll ist eine leichtgewichtige Variante zur einfachen Zusammenarbeit, zur Interaktion und zum Datenaustausch über Anwendungsgrenzen hinweg.



OpenID beschreibt die dezentrale Anmeldung an Webdiensten mittels einer URL-basierten Identität. Das offene Protokoll wurde im Jahr 2005 von Brad Fitzpatrick in der Version 1.0 entwickelt. Der Fokus für OpenID 1.0 lag vorrangig auf der Unterstützung der Authentifizierung von Nutzern und deren Registrierung. Erst in der Version 2.0, die im Dezember 2007 festgeschrieben wurde, ist der Austausch von Identitäts-Attributen mit aufgenommen worden. Im Jahr 2007 wurde ebenfalls die OpenID Foundation gegründet und übernimmt seitdem die Vermarktung von OpenID. Der aktuelle Vorschlag zur Weiterentwicklung ist OpenID Connect. OpenID wird dabei auf der Basis von OAuth 2.0 aufgesetzt, sodass die Implementierung vereinfacht und zusätzlich die Anmeldung in Desktop-Anwendungen möglich ist. Weiterhin soll zukünftig auch die eigene E-Mail-Adresse als Identität genutzt werden können.

Vorangetrieben durch Firmen wie Yahoo, Google, IBM, Myspace und Facebook gewinnt der Standard immer mehr an Verbreitung. Durch sie wird der Stan-

dard eingesetzt und die Nutzerbasis stetig erweitert. Aktuell kommt OpenID in mehr als 50.000 Websites zum Einsatz. Unterschiedliche Webportale profitieren somit von den rund einer Milliarde benannten OpenID-Identitäten der Global Player (siehe Login in Abbildung 1).

Die technischen Möglichkeiten, eine OpenID eines anderen Anbieters zu verwenden beziehungsweise eine selbst bereitzustellen, sind denkbar einfach. Für verschiedene Entwicklungsumgebungen stehen verschiedene Bibliotheken, Module oder deployfähige Apps bereit, etwa für Apache 2, C#, Coldfusion, Haskell, Java, Perl, PHP, Python, Ruby und Squeak/Smalltalk. Für die folgenden Beispiele wurde die OpenID 2.0 Java Bibliothek `openid4java` verwendet.

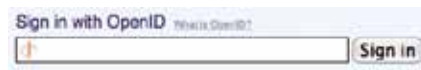


Abbildung 1: Login einer Web-Anwendung mit OpenID-Unterstützung (Quelle: <http://openid.net/>)

Anmeldung mit einer OpenID

Am OpenID-Anmeldeprozess sind der Nutzer als Akteur, identifiziert durch die OpenID, und die beiden Dienste, der OpenID-Consumer oder auch als Relying Party bezeichnet und der Identity-/OpenID-Provider, beteiligt. Der Nutzer steuert

den gesamten Anmeldeprozess. Er gibt die gültige OpenID an, meldet sich gegebenenfalls mit seiner Nutzernamen/Passwort-Kombination an, autorisiert den gewählten OpenID-Provider und bestätigt bei Bedarf die Abfrage von Nutzerattributen. Jeder Anmeldeschritt ist dabei transparent und durch den Nutzer einzeln steuerbar. Der OpenID-Consumer bezeichnet die Web-Anwendung, welche die OpenID-Anmeldung nutzen möchte. Sie fordert die Eingabe der OpenID ein, initiiert den Anmeldeprozess und überprüft dessen Ergebnis. Der OpenID-Provider stellt die OpenID URL-basiert bereit. Auf Seiten des Providers erfolgen die Validierung der Anmeldeinformation (etwa Nutzernamen/Passwort) und die Übermittlung des signierten Ergebnisses an den OpenID-Consumer.

Der ideale OpenID-Anmeldevorgang beginnt mit einer Authentifizierungsanfrage des Consumers für den Nutzer. Dieser wählt OpenID zur Anmeldung aus und gibt seine OpenID an. Der Consumer überprüft daraufhin das Format der angegebenen OpenID und startet den Discovery Prozess. Dabei wird mittels XRDS oder Link-Verweisen die OpenID nach dem OpenID-Provider Endpunkt angefragt. Der Consumer kann mit dieser Information den Anmeldevorgang starten. Dazu werden, wie im Protokoll beschrieben, Geheimnisse und Zeitstempel zwischen dem Consumer und dem



Provider ausgetauscht, um die Authentizität und Vertrauenswürdigkeit des Informationsaustausches sicherzustellen. Wird der OpenID-Provider zum ersten Mal innerhalb einer gültigen Browsersession vom Consumer zur Anmeldung verwendet, werden vom Provider die Anmelde-Informationen abgefragt. Erst wenn diese erfolgreich validiert sind, kann diese Anmeldung dem Consumer bestätigt werden. Um den Zugriff unberechtigter Web-Anwendungen auf eine aktuelle OpenID-Session zu unterbinden, kann optional eine Autorisierung des Zugriffs der anfragenden Web-Anwendung erfolgen. Die Übermittlung einer gültigen OpenID-Provider-Anmeldung erfolgt sicher signiert an den Consumer. Dieser vertraut den Rückgabewerten, führt eine Consumer-seitige Anmeldung durch und übernimmt gegebenenfalls übermittelte Profilinformationen. Der Zugriff auf geschützte Inhalte ist anschließend freigegeben (siehe Abbildung 2).

Directed Identity

Der ursprüngliche Ansatz von OpenID ging davon aus, dass jeder Nutzer seine OpenID in Form einer URL kennt. So sind beispielsweise die URLs des eigenen Blogs (<http://<myname>.livejournal.com>), einer Homepage (<www.myopenid-homepage.com>) oder eines speziellen OpenID-Providers (<myname>.myopenid.com) als OpenID möglich. Da es sich bei einer URL um eine mehr oder weniger komplexe Information handelt, welche für den Nutzer zusätzlich und zudem nur schwer zu merken ist, gibt es die Möglichkeit der Directed Identity. Dabei muss man beim Login nicht eine vollständige OpenID in Form einer URL eingeben, sondern kann einen speziellen OpenID-Provider (Google, Yahoo etc.) oder die Domain seines Accounts (google.com, yahoo.com etc.) eingeben. Der OpenID-Consumer löst abhängig von dieser Information den anzufragenden OpenID-Provider auf. Die Funktionalität Directed

Identity ist Bestandteil der OpenID-2.0-Spezifikation.

Der eigene OpenID-Provider

Der OpenID-Consumer und der OpenID-Provider stellen in einer produktiven Umgebung zwei voneinander getrennte Dienste dar und sind somit in verschiedenen Anwendungen implementiert. Für die im Folgenden aufgezeigten OpenID-Beispiele wird im ersten Schritt ein einfacher OpenID-Provider, basierend auf dem simple-openid Beispiel der openid4java Bibliothek, entwickelt. Alternativ kann an dieser Stelle auch ein gültiger OpenID-Provider, wie Google (<https://www.google.com/accounts/o8/id>), MyOpenID, CloudID oder OpenID.org, verwendet und die Implementierung eines eigenen Providers übersprungen werden.

Grundlage für den OpenID-Provider ist das „simple-openid“-Beispiel von openid4java 0.9.6. Für den Build und Start des

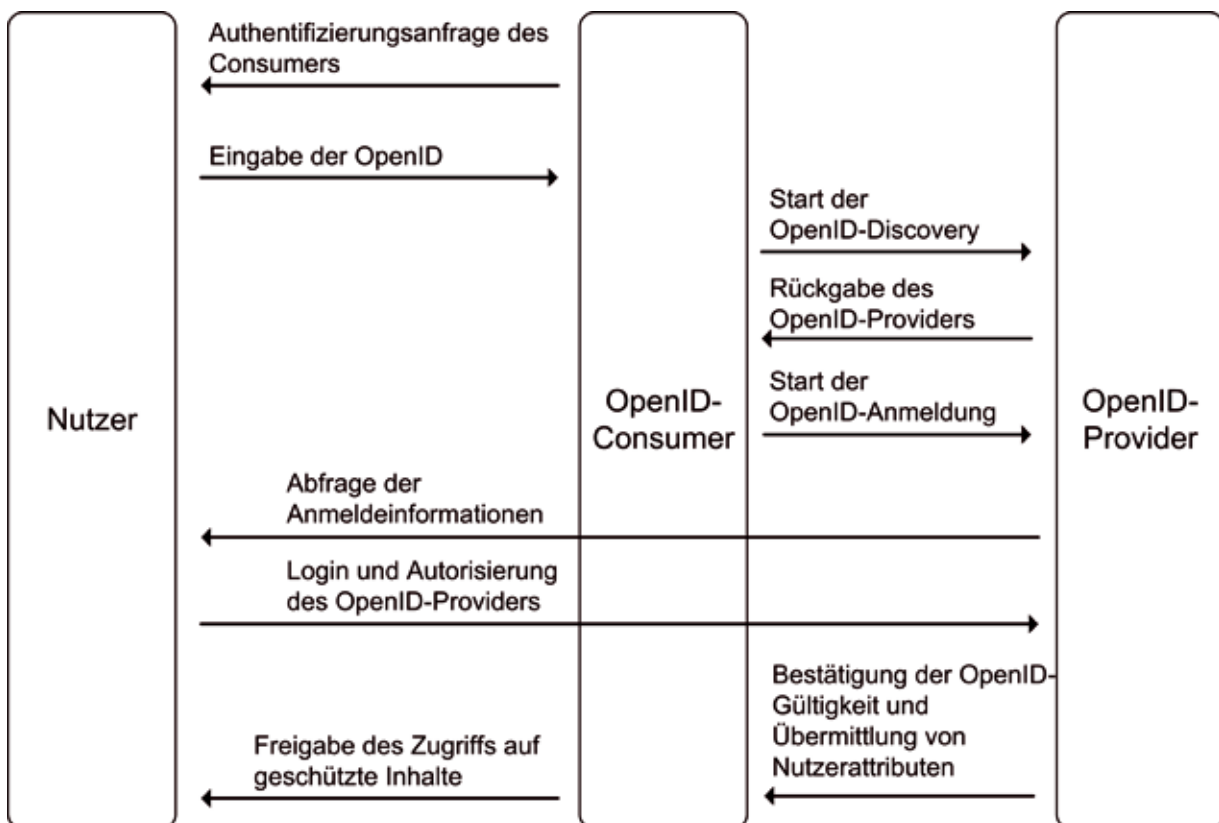


Abbildung 2: Ablauf einer OpenID-Anmeldung



Beispiels sind mindestens Java 1.4 und Maven2 erforderlich. Zum Starten der Anwendung muss ein Patch der POM erfolgen. Dazu ist die Datei „openid4java-0.9.6.662\samples\simple-openid\pom.xml“ wie in Listing1 anzupassen.

```
<dependencies>
...
</dependency>
<dependency>
  <groupId>org.openid4java</groupId>
- <artifactId>openid4java</artifactId>
+ <artifactId>openid4java-consumer</
  artifactId>
  <version>${version}</version>
+ <type>pom</type>
</dependency>
</dependencies>
```

Listing 1: POM-Patch simple-openid

Der Start der Anwendung erfolgt mit „mvn jetty:run“. Die bereitgestellte OpenID des OpenID-Providers ist über „http://localhost:8080/simple-openid/user.jsp“ aufrufbar. Als Ergebnis kommt eine XRDS-Datei zurück, welche die Url zum OpenID-Provider enthält (siehe Listing 2).

```
<?xml version="1.0" encoding="UTF-8"?>
<xrds:XRDS xmlns:xrds="xri://$xrds"
  xmlns:openid=http://openid.net/xmlns/1.0
  xmlns="xri://$xrd*(($v*2.0)(">
  <XRD>
    <Service priority="0">
      <Type>http://openid.net/signon/1.0</
        Type>
      <URI>http://localhost:8080/
simple-openid/provider.jsp</URI>
    </Service>
  </XRD>
</xrds:XRDS>
```

Listing 2: XRDS-Response einer OpenID

Der OpenID-Provider des simple-openid Beispiels prüft keine Anmelde-Informationen, sodass wir diesen dahingehend erweitern müssen. Der Link „Click To become logged in and authorize“ wird durch eine Login-Box ersetzt. Dafür ändert man die „provider_authorization.jsp“ (siehe Listing 3, Seite 49).

Damit werden beim Zugriff auf den OpenID-Provider eine Nutzernamen/Passwort-Kombination abgefragt und geprüft sowie eine vorhergehende Anmeldung

wiedererkannt. Die Überprüfung einer erfolgreichen Anmeldung am OpenID-Provider erfolgt mittels Cookie. Wurden die Anmeldeinformationen einmal erfolgreich validiert, wird das Sessioncookie „openidLoggedIn“ mit dem Wert „true“ gespeichert. Bei einer wiederholten Verwendung des OpenID-Providers wird dieses Cookie ausgewertet, sodass keine erneute Anmeldung notwendig ist (Single-signon). Ist kein openidLoggedIn-Cookie mit dem Wert „true“ gespeichert, erscheint die Login-Maske. Mit dem Nutzernamen „user“ und dem Passwort „user123“ kann man sich am OpenID-Provider anmelden. Der implementierte OpenID-Provider kann in der aufgezeigten Beispielimplementierung zur Anmeldung verwendet werden. Die relevanten Url-Endpunkte sind:

- OpenID
- <http://localhost:8080/simple-openid/user.jsp>
- OpenID-Provider
- <http://localhost:8080/simple-openid/provider.jsp>

Sicher anmelden mit OpenID

Um in einer eigenen Web-Anwendung OpenID zur Anmeldung verwenden zu können, ist ein sogenannter „OpenID-Consumer“ zu implementieren. Ausgangsbasis dafür ist das „consumer-servlet“-Beispiel der „openid4java“-Bibliothek. Wie bereits für den OpenID-Provider beschrieben, ist der Patch (siehe Listing 4) auf die Datei „openid4java-0.9.6.662\samples\consumer-servlet\pom.xml“ anzuwenden.

```
<dependencies>
...
<dependency>
  <groupId>org.openid4java</groupId>
  <artifactId>openid4java-consumer</
  artifactId>
  <version>${version}</version>
+ <type>pom</type>
</dependency>
</dependencies>
```

Listing 4: POM-Patch consumer-servlet

Die Anwendung wird mit dem Kommando „mvn -Djetty.port=7080 jetty:run“ gestartet und kann über die URL „http://localhost:7080/consumer-servlet/“ aufge-

rufen werden. Es stehen drei Beispiele zur Demonstration der OpenID-Anmeldung, Simple-Registration-Extension und Attribute-Exchange-Extension bereit.

Um die einfache OpenID-Anmeldung testen zu können, gibt man im Eingabefeld des Sample 1 die eigene OpenID (beispielsweise <http://localhost:8080/simple-openid/user.jsp>) ein. Daraufhin wird die Anmeldung vom ConsumerServlet.java initiiert (siehe Listing 5). Im ersten Schritt wird dabei der OpenID-Provider anhand der eingegebenen OpenID aufgelöst (OpenID Discovery). Mit der korrekten URL des OpenID-Providers entsteht daraufhin ein sogenannter „AuthRequest“, der per Redirect aufgerufen wird. Die relevanten OpenID-Werte werden an die URL des Providers als Parameter angehängt (siehe Listing 6).

```
List discoveries = manager.
discover(userSuppliedString);
DiscoveryInformation discovered = manager.
associate(discoveries);
...
AuthRequest authReq = manager.
authenticate(discovered, returnUrl);
...
httpResp.sendRedirect(authReq.
getDestinationUrl(true));
```

Listing 5: Initiierung der OpenID-Anmeldung

Nach Anmeldung auf Seitenn des OpenID-Providers, wird die übermittelte openid.return_to-url ergänzt um die Parameter des Providers aufgerufen, sodass die Consumer-seitige Auswertung erfolgen kann.

```
http://local.acandolocal.de:8080/simple-ope-
nid/provider.jsp?

openid.identity=http://local.acandolocal.
de:8080/simple-openid/user.jsp&
openid.return_to=http://localhost:7080/
consumer-servlet/consumer?is_
return=true&openid.rnonce=2011-07-
08T13%3A36%3A30Z0&openid.rpsig=evH0
BD001OT0dXAhyI r5yOXgI sytD7Hu6Lqhv
TLLvk0%3D&
openid.trust_root=http://localhost:7080/
consumer-servlet/consumer?is_return=true&
openid.mode=checkid_setup&
openid.ns.ext1=http://openid.net/srv/ax/1.0&
openid.ext1.mode=fetch_request
```

Listing 6: Initialer Aufruf des OpenID-Providers mit URL-Parametern



```
<h1>Login to OpenID Provider</h1>
<%
Cookie openIDCookie = null;
Cookie[] cookieList = request.getCookies();
for (Cookie cookie : cookieList) {
    if („openIDLoggedIn“.equals(cookie.getName()) && „true“.equals(cookie.getValue())) {
        openIDCookie = cookie;
    }
}

if (request.getParameter(„action“) == null && openIDCookie == null) {
    String site=(String) (openidrealm == null ? openidreturnto : openidrealm);
    if (request.getParameter(„error“) != null && !““.equals(request.getParameter(„error“))) {
        out.println(„<span style=“color:red“>“ + request.getParameter(„error“) + „</span><p>“);
    }
}
%>
<form action=“?action=authorize“ method=“POST“>
Username: <input type=“text“ name=“username“ size=“12“ maxlength=“12“><p>
Password: <input type=“password“ name=“password“ size=“12“ maxlength=“12“> <input type=“submit“ value=“Login“>
</form>
<p>
<strong>ClaimedID:</strong> <pre><%= openidclaimedid%></pre>
<strong>Identity:</strong> <pre><%= openididentity %> </pre>
<strong>Site:</strong> <pre> <%= site %></pre>
</p>

<!-- Click <a href=“?action=authorize“ id=“login“>To become logged in and authorize</a> -->
<%
} else {
    if (openIDCookie != null) {
        session.setAttribute(„authenticatedAndApproved“, Boolean.TRUE); // No need to change openid.* session vars
        response.sendRedirect(„provider.jsp?_action=complete“);
    } else if (request.getParameter(„username“) != null && request.getParameter(„password“) != null) {
        if („user“.equals((String) request.getParameter(„username“)) && „user123“.equals((String) request.getParameter(„password“))) {
            Cookie ssoCookie = new Cookie(„openIDLoggedIn“, „true“);
            response.addCookie(ssoCookie);
            session.setAttribute(„authenticatedAndApproved“, Boolean.TRUE); // No need to change openid.* session vars
            response.sendRedirect(„provider.jsp?_action=complete“);
        } else {
            response.sendRedirect(„provider_authorization.jsp?error=Wrong username/password given!“);
        }
    } else {
        response.sendRedirect(„provider_authorization.jsp?error=No username/password given!“);
    }
}
%>
```

Listing 3: Einbindung einer Login-Maske in die „provider_authorization.jsp“

Trainings für Java / Java EE

- Java Grundlagen- und Expertenkurse
- Java EE: Web-Entwicklung & EJB
- JSF, JPA, Spring, Struts
- Eclipse, Open Source
- IBM WebSphere, Portal und RAD
- Host-Grundlagen für Java Entwickler

Wissen wie´s geht

Unsere Schulungen können gerne auf Ihre individuellen Anforderungen angepasst und erweitert werden.

Weitere Themen und Informationen zu unserem Schulungs- und Beratungsangebot finden Sie unter www.aformatik.de

aformatik.[®]

aformatik Training & Consulting GmbH & Co. KG
Tilsiter Str. 6 | 71065 Sindelfingen | 07031 238070

www.aformatik.de



```
ParameterList response = new ParameterList(httpReq.getParameterMap());

DiscoveryInformation discovered = (DiscoveryInformation) httpReq.getSession().
getAttribute(„openid-disc“);
...
VerificationResult verification = manager.verify(receivingURL.toString(), response, discovered);

Identifier verified = verification.getVerifiedId();
if (verified != null) {
    ...
    return verified; // success
}
```

Listing 7: Verarbeitung des OpenID-Response, Auszug aus „ConsumerServlet.verifyResponse(„)“

```
openid.mode=checkid_setup
...
openid.ns.ax=http://openid.net/srv/ax/1.0
openid.ax.mode=fetch_request
openid.ax.type.name=http://axschema.org/namePerson
openid.ax.type.email=http://axschema.org/contact/email
openid.ax.type.web=http://axschema.org/contact/web/default
openid.ax.type.country= http://axschema.org/contact/country
```

Listing 8: OpenID-Attribute-Exchange URL-Parameter

```
FetchRequest fetch = FetchRequest.createFetchRequest();

fetch.addAttribute(„FirstName“, „http://schema.openid.net/namePerson/first“, true);
fetch.addAttribute(„LastName“, „http://schema.openid.net/namePerson/last“, true);
fetch.addAttribute(„Email“, „http://schema.openid.net/contact/email“, true);

fetch.setCount(„Email“, 1);

AuthRequest req = _consumerManager.authenticate(discovered, return_to);
req.addExtension(fetch);
```

Listing 9: Abfrage von OpenID-Nutzerattributen mittels „openid4java“

Der „org.openid4java.consumer.ConsumerManager“ verifiziert die Überprüfung der aufgerufenen OpenID-Consumer-URL (openid.return_to-Parameter). Als Parameter wird die vollständige Consumer-URL mit Querystring, die ParameterList des HTTPRequests und die DiscoveryInformation, welche zu Beginn der Initialisierung über die OpenID aufgelöst wurde, übergeben (siehe Listing 7).

Als Verified-Identifier kommt die OpenID „http://localhost:8080/simple-openid/user.jsp“ zurück.

OpenID Attribute Exchange

Das Ergebnis einer erfolgreichen OpenID-Anmeldung ist die verifizierte OpenID.

Eine zum Login angegebene OpenID wird durch den Provider bestätigt. Für viele Anwendungsfälle ist es jedoch praktisch, zusammen mit einer Identitätskennung gleichzeitig Nutzerattribute übermittelt zu bekommen. Die Anfrage vom Consumer und die Übermittlung durch den Provider dieser Daten erfolgt mittels „OpenID Attribute Exchange“. Der „checkid_setup-Request“, der prüfen soll, ob ein Nutzer Besitzer einer bestimmten OpenID ist, wird dabei um die zusätzlichen Felder ergänzt. Für einen Attribute Exchange Request werden dafür die folgenden Felder an den AuthRequest angehängt. Die angeforderten Attribute werden über die Parameter openid.ax.type.[alias] gesteuert (siehe Listing 8).

Um den FetchRequest an den AuthRequest anzuhängen, sind die Codezeilen aus dem Listing 9 zu verwenden.

Sample 3 der Beispielanwendung bietet die Möglichkeit, den Anwendungsfall „OpenID Attribute Exchange“ nachzuvollziehen. Als OpenID sollte dafür eine Identitäts-URL eines OpenID-Providers in der Version 2.0, wie die Google-OpenID <https://www.google.com/accounts/o8/id> zum Einsatz kommen.

Fazit

OpenID ist als Standard zum Plattformübergreifenden Single-sign-on in Web-Anwendungen kaum noch wegzudenken. Insbesondere seine Einfachheit hinsichtlich des Protokolls als auch dessen Implementierung zeichnet OpenID gegenüber SAML aus. Der Anwendungsentwickler erhält durch eine Vielzahl an Bibliotheken die Fähigkeit, schnell und unkompliziert den Protokoll-Stack zu implementieren und seine Anwendung OpenID-fähig zu machen.

Als Web-Anwendung mit kleiner Nutzerbasis ermöglicht es OpenID ohne viel Aufwand, neue Anwender für ein Online-Produkt zu gewinnen. Nutzer können sich mit einer bestehenden Kennung (Google, Facebook etc.) an der eigenen Anwendung anmelden, ohne einen umfangreichen Registrierungsprozess zu durchlaufen oder sich eine neue Nutzernamen/Passwort-Kombination ausdenken zu müssen. Zusätzlich werden Profildaten nur an einer Stelle hinterlegt und können gegebenenfalls aktualisiert beziehungsweise gesperrt werden.

Sebastian Glandien
sebastian.glandien@acando.de

Sebastian Glandien ist IT-Consultant bei der Acando GmbH mit dem Schwerpunkt Java-Enterprise-Anwendungen. Das besondere persönliche Interesse von ihm gilt den Themen „Identity- und Access-Management“, „Web-Service Security“ und „mobile Plattformen“.





Bestellen eines kostenlosen Exemplares der Zeitschrift Java aktuell

Anschrift:

Name, Vorname

Firma

Abteilung

Straße, Hausnummer

PLZ, Ort

ggf. Rechnungsanschrift

E-Mail

Telefonnummer

Die allgemeinen Geschäftsbedingungen* erkenne ich an, Datum, Unterschrift

Jetzt Abonnement sichern:

- Abonnement Newsletter: Java aktuell – der iJUG-Newsletter, kostenfrei
- Java aktuell – das iJUG-Magazin Abo: vier Ausgaben zu 18 Euro im Jahr

Für Oracle-Anwender und Interessierte gibt es das Java aktuell Abonnement auch mit zusätzlich sechs Ausgaben im Jahr der Fachzeitschrift DOAG News und zwei Ausgaben im Jahr Business News. Weitere Informationen unter www.doag.org/shop/

Senden Sie das ausgefüllte Formular an:

Interessenverbund der Java User Groups e.V.
Tempelhofer Weg 64
12347 Berlin

oder faxen Sie es an:

0700 11 36 24 39

oder bestellen Sie online:

go.ijug.eu/go/abo

*Allgemeine Geschäftsbedingungen:

Zum Preis von 18 Euro (inkl. MwSt.) pro Kalenderjahr erhalten Sie vier Ausgaben der Zeitschrift "Java aktuell - das iJUG-Magazin" direkt nach Erscheinen per Post zugeschickt. Die Abonnementgebühr wird jeweils im Januar für ein Jahr fällig. Sie erhalten eine entsprechende Rechnung. Abonnementverträge, die während eines Jahres beginnen, werden mit 4,90 Euro (inkl. MwSt.) je volles Quartal berechnet. Das Abonnement verlängert sich automatisch um ein weiteres Jahr, wenn es nicht bis zum 31. Oktober eines Jahres schriftlich gekündigt wird. Die Widerrufsfrist beträgt 14 Tage ab Vertragserklärung in Textform ohne Angabe von Gründen.

Impressum

Herausgeber:
Interessenverbund der Java User
Groups e.V. (iJUG)
Tempelhofer Weg 64, 12347 Berlin
Tel.: 0700 11 36 24 38
www.ijug.eu

Verlag:
DOAG Dienstleistungen GmbH
Fried Saacke, Geschäftsführer
info@doag-dienstleistungen.de

Chefredakteur (VisdP):
Wolfgang Taschner,
redaktion@ijug.eu

Chefin von Dienst (CvD):
Carmen Al-Youssef,
office@ijug.eu

Titel, Gestaltung und Satz:
Claudia Wagner,
DOAG Dienstleistungen GmbH

Anzeigen:
CrossMarkeTeam, Ralf Rutkat,
Doris Budwill
redaktion@ijug.eu

Mediadaten und Preise:
[http://www.ijug.eu/images/
vorlagen/2011-ijug-mediadaten_
java_aktuell.pdf](http://www.ijug.eu/images/vorlagen/2011-ijug-mediadaten_java_aktuell.pdf)

Druck:
adame Advertising and Media
GmbH Berlin
www.adame.de

Java aktuell – das Abo
4 Ausgaben für 18 Euro