

Hier wird „Service“ groß geschrieben

Die OSGi Service Platform (OSGi) hat sich zu einem sehr bedeutenden Standard im Java-Umfeld entwickelt. Also wird es für den engagierten Java-Entwickler allerhöchste Zeit, sich damit näher auseinander zu setzen.



von Heiko Seeberger

Die ersten drei Artikel dieser Serie (Kasten: „Artikelserie: OSGi in kleinen Dosen“) über die technischen Grundlagen von OSGi beleuchteten ausführlich das OSGi-Framework, das die Basis für Modularisierung, Laufzeitdynamik und Services in der Java-Welt darstellt. Im letzten Artikel lernten wir mit Declarative Services bereits einen sehr wichtigen Vertreter der so genannten Standardservices

kennen. Heute werden wir zwei weitere Standardservices unter die Lupe nehmen und damit diese Serie abschließen.

Frameworkservices und Service Compendium

Was ist eigentlich ein OSGi-Standard-service? Die Antwort auf diese Frage gibt einen Blick in die OSGi-Spezifikation [1]. Dort finden wir zum einen – in der Core Specification – fünf so genannte

Frameworkservices: Package Admin Service, Start Level Service, (Conditional) Permission Admin Service und URL Handlers Service. Diese bieten optional eine Erweiterung der Funktionalität des OSGi-Frameworks an, z. B. eine Möglichkeit zur Festlegung der Startreihenfolge von Bundles oder zur zentralen Verwaltung der Permissions von Bundles. Zum anderen finden wir im Service Compendium eine Vielzahl

Quellcode
auf CD

von Spezifikationen für spezifische horizontale Einsatzbereiche (Kasten: „Service Compendium“).

Nicht alle dieser Spezifikationen sind Services im eigentlichen Sinn, z. B. der bereits im dritten Artikel vorgestellte Service Tracker oder die Initial Provisioning Specification. Des Weiteren fällt auf, dass OSGi seine Herkunft nicht leugnen kann: Wir finden zahlreiche Spezifikationen für den Bereich embedded/mobile. Wir werden im Folgenden zwei Standardservices aus dem Service Compendium betrachten, die sich insbesondere gut für den Einsatz im Enterprise-Umfeld eignen.

Log Service

Kaum ein Softwaresystem kommt ohne Logging aus.

Die besondere Herausforderung beim Logging in der OSGi-Welt besteht darin, dass die Logging-Nachrichten dezentral – und zwar in den einzelnen Bundles – erzeugt werden, in den meisten Fällen jedoch zentral geschrieben werden sollen, z. B. in eine zentrale Logging-Datei. OSGi begegnet dieser Herausforderung mit der Log Service Specification [2]. Darin wird ein zweigeteiltes API spezifiziert: Das Interface `LogService` dient zum Schreiben von Logging-Nachrichten und das Interface `LogReaderService` bietet die Möglichkeit, geschriebene Logging-Nachrichten zu lesen oder `LogListener` zu registrieren, die die Logging-Nachrichten beim Schreiben erhalten.

Wer schon einmal mit einem der gängigen Logging-Frameworks wie Log4j oder SLF4J gearbeitet hat, wird mit dem `LogService` rasch zurechtkommen. Schließlich bietet er ein im

Vergleich zu den genannten Lösungen recht abgespecktes API: Logging-Nachrichten können in verschiedenen Levels geschrieben werden, wobei neben der Nachricht selbst optional ein `Throwable` und/oder eine `ServiceReference` übergeben werden können. Letzteres ist die einzige OSGi-spezifische Besonderheit, ansonsten fehlen einige der sonst gewohnten Features, z. B. die Überprüfung, ob ein gewisser Level überhaupt „aktiviert“ ist (`isDebugEnabled()`). Zur Nutzung des Log Service wird eine Im-

In der OSGi-Welt bedeutet Logging, dass Nachrichten dezentral erzeugt werden.

plementierung der Log Service Specification benötigt. Wir verwenden für unser Beispiel die Equinox-Implementierung, die durch das `Bundle.org.eclipse.equinox.log` bereitgestellt wird. Diese Implementierung bietet eine Integration in die Equinox Console: Mit dem Kommando `log` können die letzten Logging-Nachrichten angezeigt werden, wobei unter Angabe der Bundle-ID als Parameter die Logging-Nachrichten auf das entsprechende Bundle eingeschränkt werden können (Abb. 1).

Da es sich beim Log Service um einen OSGi-Service handelt, gilt es, bei der Benutzung die üblichen Spielregeln zu berücksichtigen: Der Log Service kann schließlich „kommen und gehen“. Hierfür bietet sich die Verwendung der im letzten Artikel beschriebenen Declarative Services an. In unserem Beispiel modifizieren wir die Shell Component derart, dass sie neben den `ContactRepositories` auch einen `LogService` referenziert (Listing 1). Da wir diese Referenz „mandatory“ machen, wird die Komponente nur aktiv, wenn ein `LogService` zur Verfügung steht, sodass wir uns bei dessen Verwendung keine Sorgen machen müssen (d. h. die Referenz wird niemals null sein, Listing 2).

Der Log Service meistert zwar die eingangs angesprochene Herausforderung, dezentrale Logging-Nachrichten zentral zu bündeln, kann aber keineswegs als alleiniger Heilsbringer bezeichnet

```
osgi> log 7
>Info [7] BundleEvent STARTED ini
>Info [7] Activating ... initial@
>Info [7] ServiceEvent REGISTERED
```

Abb. 1: Equinox Log Service und Equinox Console

werden. Zunächst fehlen einige Features, die andere Logging-Lösungen längst bieten, z. B. die Aktivierung von Levels und deren Überprüfung per API oder die Möglichkeit, den Transport von Logging-Nachrichten an `LogListener` anhand von Kriterien einzuschränken. Aber das eigentliche Problem besteht darin, dass „echte“ Systeme oft zahlreiche Libraries einsetzen, die wiederum eine bestimmte Logging-Lösung wie Log4j oder SLF4J verwenden und

eben nicht den OSGi Log Service. Somit kann man zwar die Logging-Nachrichten der eigenen Bundles durch die Verwendung des Log Service zentralisieren, jedoch wird man die Logging-Nachrichten von „fremden“ Bundles in separaten Kanälen finden. Doch gibt es kein Problem ohne Lösung: Die meisten der gängigen Logging-Lösungen bieten ein Design mit einem API und eine separate Implementierung, sodass deren Standardver-

Service Compendium

- Log Service
- Http Service
- Device Access
- Configuration Admin Service
- Metatype Service
- Preferences Service
- User Admin Service
- Wire Admin Service
- IO Connector Service
- Initial Provisioning
- UPnP Device Service
- Declarative Services
- Event Admin Service
- Deployment Admin
- Auto Configuration
- Application Admin Service
- DMT Admin Service
- Monitor Admin Service
- Foreign Application Access
- Service Tracker
- XML Parser Service
- Position
- Measurement and State
- Execution Environment

Artikelserie: OSGi in kleinen Dosen

Teil 1: Erste Schritte mit OSGi

Teil 2: Immer in Bewegung – Bundles und Lifecycle

Teil 3: Was wünschen Sie? – Services à la OSGi

Teil 4: Services auf deklarative Weise

Teil 5: Hier wird „Service“ groß geschrieben



Abb. 2: HelloServlet à la OSGi

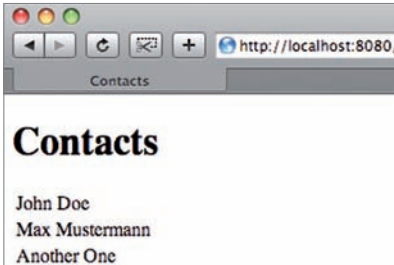


Abb. 3: Ausgabe aller Contacts mit Servlet und JSP

halten modifiziert werden kann. Eine aus unserer Sicht besonders gelungene Lösung dieses „Multi-API-Logging“ bietet PAX Logging [3]: Das API Bundle bietet das API der meisten gängigen Logging-Lösungen einschließlich dem OSGi Log

Service an. Die Logging-Nachrichten werden zentral verarbeitet, egal über welches API sie erzeugt wurden, sodass eine einheitliche Konfiguration und vor allem ein zentrales Logging aller Bundles erzielt werden können.

Http Service

OSGi erobert derzeit, nach der embedded/mobile-Welt und dem Desktop, nun auch den Server. Dazu passend stellen wir nun den Http Service [4] vor. Dieser ist in der Standardversion gar nicht so aktuell, schließlich basiert er nur auf Version 2.1 des Servlet API und bietet auch keine Unterstützung für JavaServer Pages. Das Interface *HttpService* ermöglicht das Registrieren von statischen Ressourcen und Servlets. Dabei wird neben der Ressource oder dem Servlet im Wesentlichen ein so genannter Alias übergeben, der dem Servlet Path der URL entspricht. Listing 3 zeigt den Ausschnitt einer Service Component, die bei der Aktivierung unter dem Alias */hello* ein *HelloServlet* registriert und die Registrierung bei der Deaktivierung wieder aufhebt. Dadurch kann es unter dem URL *http://localhost:8080/hello* (Host und Port können natürlich variieren) aufgerufen werden. In unserem Beispiel resultiert das in einfachem statischem HTML-Code (Abb. 2).

Natürlich wird der Http Service in dieser Standardversion OSGi nicht zum entscheidenden Durchbruch auf dem Server verhelfen. Dazu fehlen einfach zu viele Features, die von heutigen Web-Applications vorausgesetzt werden, z. B. Servlet Filter und JavaServer Pages. Zum Glück gibt es einige Erweiterungen, die hier aushelfen. So bietet z. B. Equinox neben einer Standardimplementierung auch eine Variante auf Basis von Jetty und Jasper an, die Servlet API 2.4 und JavaServer Pages 2.0 unterstützt. Noch aktueller ist derzeit PAX Web [5], ebenfalls mit gebündeltem Jetty und Support für Servlet API 2.5 und JavaServer Pages 2.1.

In unserem Beispiel verwenden wir PAX Web, um gemäß Model-2-Design mit einem Servlet die Kontakte aller *ContactRepositories* abzufragen und diese mit einer JSP unter Verwendung von Taglibs anzuzeigen (Abb. 3). Die resul-

tierende Verzeichnisstruktur entspricht dank eines speziellen *HttpContext*, der den Servlet Path zu einer Bundle-Ressource auflöst, der Standardstruktur für Web-Applications, d. h. die JSPs, Libraries und TLDs liegen in einem Verzeichnis *webapp* mit Unterverzeichnis *WEB-INF*.

Fazit

Die beiden exemplarisch vorgestellten OSGi-Standardservices zeigen, dass OSGi nicht nur ein abstraktes dynamisches Modulsystem für Java bietet, sondern darüber hinaus wichtige Einsatzszenarien durch horizontale Dienste abdeckt. Natürlich muss die Spezifikation die Geschwindigkeit des technologischen Wandels mit aufnehmen, beispielsweise beim Versionsstand des Http Service. Wer die aktuelle Entwicklung der im Sommer dieses Jahres erwarteten nächsten Version 4.2 verfolgt, wird feststellen, dass sich gerade im Bereich der Enterprise-Technologien auch tatsächlich eine ganze Menge tut: Transaktionen, Verteilung und Blueprint Service, auch bekannt als Spring Dynamic Modules. Wir dürfen gespannt sein! ■



Heiko Seeberger ist als Technical Director für die Weigle Wilczek GmbH tätig. Sein technischer Schwerpunkt liegt in der Entwicklung von Unternehmensanwendungen mit OSGi, Eclipse RCP, Spring, AspectJ und Java EE. Seine Erfahrungen aus über zehn Jahren IT-Beratung und Softwareentwicklung fließen in die Eclipse Training Alliance ein. Zudem ist Heiko Seeberger aktiver Committer in Eclipse-Projekten, Autor zahlreicher Fachartikel und Redner auf einschlägigen Konferenzen.

Listing 1

```
<component name="com.weiglewilczek...shell">
<implementation class="com.weiglewilczek...Component"/>
<service>
<provide interface="org.eclipse...CommandProvider"/>
</service>
<reference name="contactRepositories"
interface="com.weiglewilczek...ContactRepository"
cardinality="1..n"/>
<reference name="logService"
interface="org.osgi...LogService"
cardinality="1..1"/>
</component>
```

Listing 2

```
protected void activate(ComponentContext context) {
logService = (LogService) context.locateService("logService");
logService.log(LogService.LOG_INFO, "Activating ...");
....
}
```

Listing 3

```
private static final String HELLO = "/hello";

protected void activate(ComponentContext context) throws Exception {
httpService = (HttpService) context.locateService("httpService");
httpService.registerServlet(HELLO, new HelloServlet(), null, null);
}

protected void deactivate(ComponentContext context) {
httpService.unregister(HELLO);
}
```

Links & Literatur

- [1] OSGi-Spezifikation: www.osgi.org/Specifications/
- [2] OSGi Service Compendium, S. 5ff, Log Service Specification
- [3] PAX Logging: wiki.ops4j.org/display/ops4j/Pax+Logging
- [4] OSGi Service Compendium, S. 17 ff., Http Service Specification
- [5] PAX Web: wiki.ops4j.org/display/ops4j/Pax+Web