

## RESTlos glücklich

Wer bisher mithilfe von Java versuchte, dem Ansatz von REST-basierten Services zu folgen, sah sich zuweilen ziemlich alleingelassen. Mit dem Ende 2008 unter dem Titel „JAX-RS“ verabschiedeten JSR 311 wurde dieser Umstand inzwischen beseitigt. Dem Java-Entwickler soll mit JAX-RS ein standardisiertes Programmiermodell an die Hand gegeben werden, das die Umsetzung von RESTful Web Services doch deutlich vereinfacht. Der Umstand, dass JAX-RS nun offizieller Bestandteil der neuen Java Enterprise Edition 6 ist, soll Anlass sein, den Standard einmal genauer zu betrachten.

Die Aufgabe von JAX-RS ist schnell umschrieben: Es gilt ein einfaches Programmiermodell bereitzustellen, mit dessen Hilfe unterschiedliche Repräsentationen von Ressourcen über URIs publiziert werden. Der Zugriff auf diese Ressourcen muss über einen für REST typischen Standardsatz von Operationen erfolgen. Die Antwort von JAX-RS auf diesen Anforderungskanon ist die Abbildung einzelner Aspekte des HTTP-Protokolls auf entsprechende Java-Konstrukte. JAX-RS ist also die Java-basierte Umsetzung von REST auf der Basis von HTTP. Eine Umsetzung hinsichtlich anderer Protokolle ist nicht vorgesehen. Sie erscheint aufgrund der breiten Akzeptanz, REST-Services über HTTP umzusetzen, auch nicht notwendig. Anzumerken bleibt nur, dass der Fokus der Spezifikation ausschließlich

auf der Betrachtung serverseitiger Aspekte liegt. Belange der Client-Seite werden nicht aufgegriffen und müssen somit auch weiterhin vollumfänglich in der Anwendungslogik behandelt werden.

### RESTlos einfach

Das Publizieren von Ressourcen, wie auch ein Großteil der angrenzenden Aspekte erfolgt mit JAX-RS über einen umfangreichen Satz verschiedener Annotationen. Um die unterschiedlichen Blickwinkel des Standards zu illustrieren, werden wir auf Code-Fragmente einer fiktiven Kalenderanwendung zurückgreifen. Starten wir also mit dem eigentlichen Rückgrat jeder REST-Anwendung: Der Publikati-

on von Ressourcen über einen URI. Eine Ressource im Sinne von JAX-RS ist ein einfaches Java-Objekt. Für dieses muss mithilfe der Annotation `@Path` ein Kontextpfad zum späteren Zugriff über eine URL definiert werden (Abbildung 1).

```
@Path("/calendar/entry")
public class CalendarResource {
    //...
}
```

Abb. 1: Ressourcenklasse

Bei der Angabe des Pfades gibt es einige wichtige Gesichtspunkte zu beachten: Die Angabe des Pfades erfolgt relativ. Der vollständige Pfad ergibt sich erst in Abhängigkeit des konkreten Anwendungsdeployments. Weiterhin räumt JAX-RS die Möglichkeit ein, auch variable Anteile innerhalb des Kontextpfades zu definieren. Dies spielt u. a. dann eine Rolle, wenn über den URI gewissermaßen auch Daten (z. B. der Primärschlüssel eines Kalendertermins) transportiert werden. Die variablen Anteile können in Form von URI-Templates (in unserem Beispiel z. B. der URI-Anteil, der später einmal die ID eines Kalendereintrags enthalten wird), aber auch unter Verwendung von regulären Ausdrücken formuliert werden (Abbildung 2).

```
@Path("/calendar/entry/{id}")
public class CalendarResource {
    //...
}
```

Abb. 2: Ressourcenklasse mit Pfad-Parameter

HTTP-Operationen	JAX-RS Annotationen	Bedeutung
GET	@GET	Auslesen einer Ressource
POST	@POST	Erzeugen einer neuen Ressource
PUT	@PUT	Update auf einer bereits bestehenden Ressource
DELETE	@DELETE	Löschen einer Ressource
HEAD	@HEAD	Auslesen einer Ressource; im HTTP-Response werden jedoch keine Nutzdaten übertragen
OPTIONS	@OPTIONS	Extraktion des Methodenparameters aus einem URI; ermittelt die an einer Ressource zulässigen HTTP-Operationen

Tabelle 1: HTTP-Operationen

Annotationen	Bedeutung
@PathParam	Extraktion des Methodenparameters aus dem URI.
@QueryParam	Extraktion des Methodenparameters aus einem URI-Query-Parameter.
@FormParam	Extraktion des Methodenparameters aus einem Formparameter.
@CookieParam	Extraktion des Methodenparameters aus einem HTTP-Cookie.
@HeaderParam	Extraktion des Matrixparameters aus einem HTTP-Header.
@MatrixParam	Extraktion des Methodenparameters aus einem URI-Matrixparameter.
@Default	Angabe eines Standardwertes für den Fall, dass der Parameter nicht an der definierten Stelle im Request enthalten ist.

Tabelle 2: Zugriff auf Request-Informationen

Im nächsten Schritt auf dem Weg zum ersten fertigen REST-Service gilt es eine Abbildung zwischen den Operationen des HTTP-Protokolls und den Methoden der betreffenden Java-Klasse zu definieren. Dabei bedient man sich wieder aus dem Fundus der Annotationen von JAX-RS.

Für jede relevante HTTP-Operation existiert eine entsprechende Annotation (Tabelle 1). Mit ihnen können nun beliebige Methoden einer Java-Klasse als REST-Operation gekennzeichnet werden. Das Mapping zwischen HTTP und dem Aufruf passender Java-Methoden übernimmt die JAX-RS-Laufzeitumgebung. In unserem Beispiel wird z. B. mithilfe der Methode `getEntry` ein Kalendereintrag an den Client gesendet, wenn dieser zuvor über ein HTTP-GET am Service angefordert wurde (Abbildung 3).

```
@Path("/calendar/entry")
public class CalendarResource {
    @GET
    public String getEntry() {
        return "...";
    }
}
```

Abb. 3: Ressourcenmethode

Methoden einer Ressource, für die die JAX-RS-Laufzeitumgebung eine Abbildung auf korrespondierende HTTP-Operationen durchführt, können natürlich auch Eingabeparameter besitzen. Durch die Annotation der betreffenden Methodenparameter erfolgt zur Laufzeit automatisch eine passende Wertzuweisung (Tabelle 2 und Abbildung 4).

```
@Path("/calendar/entry/{Id}")
public class CalendarResource {
    //...
    @GET
    public void getEntry(
        @PathParam("Id")
        String entryId) {
        //...
    }
}
```

Abb. 4: Zugriff auf Request-Informationen

## RESTlos vielgestaltig

Ein Grundgedanke von REST ist die Agnostik gegenüber dem Format einer Ressourcen-Repräsentation. Im Unterschied zu anderen Konzepten besitzt REST explizit keine Schicht auf der standardisierte oder gar einem Protokoll folgende Nachrichten ausgetauscht werden. Ein attraktiver Aspekt des REST-Ansatzes

ist daher die Nutzung dieses Freiheitsgrades, um ein und dieselbe Ressource (nahezu) transparent in unterschiedlichen Formaten zu veröffentlichen. In unserem Kalender wäre z. B. die Kodierung von Terminen in verschiedenen Kalenderformaten denkbar, um ein breites Spektrum kompatibler Client-Anwendungen zu erzielen.

Mit JAX-RS gibt es ein standardisiertes Vorgehen, wie die unterschiedlichen Repräsentationen einer Ressource innerhalb einer REST-basierten Java-Anwendung definiert und verarbeitet werden. Mithilfe der Annotationen `@Provides` und `@Consumes` werden zunächst sowohl die angebotenen als auch die exportierten Formate über einen MIME-Type definiert. Im Fall unserer Terminressource sind das die in **Abbildung 5** dargestellten MIME-Typen „text/calendar“ und „application/calendar+xml“.

Wirft man einen Blick auf das Beispiel, so fällt auf, dass an unserer Terminressource nun eine ganze Reihe verschiedener Methoden auftauchen, die sich um die Erzeugung des korrekten Formates kümmern. Um diesem Wildwuchs entgegenzuwirken und sinnvollerweise die Erzeugung gängiger Formate in einer separaten Logik auszulagern, führt JAX-RS das Konzept so genannter Entity-Provider ein. Durch die Implementierung der Interfaces `MessageBodyWriter` oder `MessageBodyReader` ist man in der Lage, für einen definierten MIME-Typen auch eine zentrale und einheitliche Implementierung für die Konvertierung zwischen Java-Objekten und dem spezifischen Format zu realisieren.

Durch die Annotation `@Provider` ist eine JAX-RS-Laufzeitumgebung in der Lage, die im Klassenpfad verfügbaren Entity-Provider zu identifizieren, so dass eine explizite Registrierung nicht notwendig

ist. Die Identifikation für die korrekte Abbildung der MIME-Typen erfolgt mithilfe der HTTP-Header Content-Type für das Request-Mapping sowie Accept für das Response-Mapping (Abbildung 6).

Adäquat zu dem `MessageBodyWriter` wäre auch ein passender `MessageBodyReader` möglich, um auf der Request-Seite ein zentrales Typmapping umzusetzen. Vorausgesetzt, wir hätten einen solchen `MessageBodyReader` implementiert, wäre z. B. eine Methode zum Update unserer Terminressource, wie in [Abbildung 7](#) dargestellt, zu realisieren. Die Abbildung zwischen dem Format des HTTP-Request-Bodies und dem Entity-Parameter vom Typ `CalendarEntry` wäre von nun an für die Anwendung transparent.

## RESTlos hierarchisch

Bei genauer Betrachtung des Terminbeispiels wird schnell deutlich, dass bislang ein wesentlicher Bestandteil außen vor gelassen wurde. Bisher fehlt den veröffentlichten Ressourcen noch jegliche hierarchische Struktur. Ein Termin kann z. B. einem Kalender zugeordnet werden. JAX-RS unterstützt den Entwickler bei der Umsetzung solcher hierarchischer Ressourcengefüge mithilfe einer Konvention für die Methoden einer Ressource. Methoden, welche eine Annotation für das HTTP-Operationsmapping tragen (z. B. `@GET`) und zusätzlich auch mit einem Subkontext über `@Path`

```
@Path("/calendar/entry/{entryId}")
public class CalendarResource {

    @GET
    @Produces("text/calendar")
    public String getEntryAsICal() {
        return "BEGIN:VCALENDAR ...";
    }

    @GET
    @Produces("application/calendar+xml")
    public String getEntryAsXCal() {
        return "<ical:vcalendar iTIP:method='publish'>...";
    }
}
```

Abb. 5: Beispiel für unterschiedliche Kalenderformate

```
@Provider
@Produces("text/calendar")
public class ICalMessageBodyWriter
    implements MessageBodyWriter<CalendarEntry> {

    public long getSize(CalendarEntry cEntry, Class<?> cEntryClazz,
        Type t, Annotation[] annotations,
        MediaType mt) {
        return -1; //Keine Information zur Größe
    }

    public boolean isWriteable(Class<?> cEntryClazz, Type t,
        Annotation[] annotations,
        MediaType mt) {
        return true;
    }

    public void writeTo(CalendarEntry cEntry, Class<?> cEntryClazz,
        Type t, Annotation[] annotations,
        MediaType mt,
        MultivaluedMap<String, Object> httpHeaders,
        OutputStream outputStream)
        throws IOException, WebApplicationException {
        // Code zur Formatkonvertierung
    }
}
```

Abb. 6: Producer für „text/calendar“

versehen sind, bezeichnet der Standard als Sub-Resource-Methoden. Gemeint ist damit eine Logik, Ressourcenanfragen gezielt durch diese Methoden bedienen zu lassen.

Fehlt einer Methode mit Pfadangabe die Annotation für das Operationsmapping, spricht der Standard von so genannten Sub-Resource-Locators. Diese besitzen keine Entity-Parameter und geben lediglich eine Instanz der Sub-Resource zurück, welche den ursprünglichen Request dann verarbeitet. Bezogen auf das Terminbeispiel heißt das, dass Requests, die über den Kalender-URI für einen bestimmten Termin eingehen, durch das betreffende Terminobjekt verarbeitet werden ([Abbildung 8](#)).

```
@Path("/calendar/entry/{id}")
@Consumes({"text/calendar", "application/calendar+xml"})
public class CalendarResource {

    @PUT
    public void updateEntry(@PathParam("id") String id,
        CalendarEntry entry) {
        //Aktualisieren eines Kalendereintrags
    }
}
```

Abb. 7: Ressourcenklasse mit `@Consumes`-Annotation

```
@Path("/calendar")
public class CalendarResource {

    @Path("entry/{id}")
    public CalendarResource findCalendarEntry
        (@PathParam("id") String id) {
        CalendarEntryResource cer = new CalendarEntryResource(id);
        return cer;
    }

    //Sub-Ressource
    public class CalendarEntryResource {
        @PersistenceContext
        private EntityManager em;
        private String id;
        public CalendarEntryResource(String id) {
            this.id = id;
        }

        @GET
        @Produces("application/calendar+xml")
        public CalendarEntry getCalendarEntry() {
            // Laden eines Kalendereintrags
            CalendarEntry ce = em.find(CalendarEntry.class, this.id);
            return ce;
        }
    }
}
```

Abb. 8: Beispiel für Ressourcen-Hierarchie

## RESTlos verbunden

Ein weiterer Aspekt des REST-Paradigmas ist die clientseitige Navigation von einer Ressource zur nächsten durch gegenseitige Verlinkung. Im Idealfall könnte eine Client-Anwendung so von einer Ressource zur nächsten „browsen“. JAX-RS müsste dazu die Möglichkeit vorsehen, dynamisch generierte URI-Verlinkungen in die Antwortnachrichten eines REST-Services einzubetten. Diese Unterstützung existiert zwar, kommt jedoch reichlich gestelzt daher. Das Werkzeug für die dynamische Generierung von URIs ist die Klasse `UriBuilder`. Eine Instanz dieser Klasse bekommt

man entweder über den aktuellen Deploymentkontext (injiziert als Instanz von `UriInfo` mithilfe der `@Context`-Annotation) oder aber über eine der zahlreichen statischen Methoden, die `UriBuilder` bereits mitbringt. Im Anschluss kann man sich den gewünschten Pfad über String-Referenzen zu bestimmten URI-Template-Elementen etc. im wahrsten Sinne des Wortes zusammenbasteln (Abbildung 9).

```
@Context
UriInfo uinfo;

UriBuilder uBuilder = uinfo.getBaseUriBuilder();

URI uri = uBuilder.path(CalendarResource.class,
    "findCalendarEntry").build("123");
```

Abb. 9: Generierung von Ressourcen-URIs

## RESTvoll durchstarten

Wie gut ist also die Unterstützung für die Umsetzung von REST-Services auf Basis der Java-Plattform nun mit JAX-RS? Dem ersten Aufschlag kann man guten Gewissens das Prädikat „Praxis-tauglich“ ausstellen. Das mit der Spezifikation eingeführte Programmiermodell ist ohne Schnörkel und hält für den Entwickler kaum Fallstricke bereit. Sicher könnte man sich an der ein oder anderen Stelle einiges mehr vorstellen. So bleiben Sicherheitsaspekte in der Spezifikation nahezu ausgeblendet und mit dem Verweis, dass Lifecycle-Aspekte nicht Gegenstand der Spezifikation sind, macht man es sich doch ein wenig zu einfach. Der fehlenden Client-Integration dürfte noch so manch einer nachtrauern, wenn er wie aktuell notwendig dafür viel eigenen Code produzieren muss. Mit Erscheinen der Java Enterprise Edition 6 wird das im November 2009 verabschiedete Maintenance Release 1.1 die bislang existente Version 1.0 ablösen. Dieses enthält Vorgaben zur Integration in Java EE 6, z.B. die Publikation von Stateless Session Beans als REST-Services. Dennoch, der Versuch, dem Themengebiet REST innerhalb der Java-Plattform mehr Aussagekraft zu verleihen, ist durchaus gelungen, auch wenn schon jetzt Potenzial für eine weitere Auflage von JAX-RS ausgemacht werden kann.

AUTOR: TOBIAS ISRAEL

## kontakt

buschmais GbR  
Leipziger Straße 93  
01127 Dresden

Tel +49 (0)351 3209230  
Fax +49 (0)351 32092329  
info@buschmais.com  
www.buschmais.de