

RESTful Web Services mit JAX-RS

Überblick

- Was sind RESTful Web Services?
- JAX-RS: The Java API for RESTful Web Services.
- Jersey: Die Referenzimplementation von JAX-RS
- JAX-RS Entwicklungsprozess
 - Ressourceklassen
 - Annotationen
 - Entity Provider
 - ResponseBuilder

Was sind RESTful Web Services?

- Representational State Transfer
- Alles hat einen eindeutigen Identifier
- Alles miteinander Verlinken
- Zugriffe erfolgen mit "Standard Methoden"
- Unterschiedliche Repräsentationen
- Stateless

Eindeutige Identifier

- Alles wird mit einer ID versehen
- Alles ist über URIs erreichbar
- Beispiele:

<http://example.org/kunden/121>

<http://example.org/produkte/p4321>

<http://example.org/bestellungen/200909300001>

Alles Verlinken

- Ressourcen verweisen aufeinander

```
<bestellung nr='200909300001'>  
  <menge>2</menge>  
  <produkt ref='http://example.org/produkte/4321' />  
  <customer ref='http://example.org/kunden/121' />  
</bestellung>
```

Standard Methoden

- Methoden durch HTTP definiert
 - GET: Zugriff/Download einer Ressource
 - PUT: Ressource erstellen oder updaten
 - POST: Ressource erstellen oder Operation ausführen
 - DELETE: Ressource löschen

Unterschiedliche Repräsentationen

- Respräsentation einer Ressource abhängig vom Client
- Content Negotiation über Accept Header oder URI
- Üblich: XML, (X)HTML, JSON
- aber auch z.B. vcard, ical/vcal oder plain text

Stateless

- Persistente IDs
- Alles notwendige im Request enthalten
- Sessions vermeiden

JAX-RS

- Definiert durch JSR-311
 - <https://jsr311.dev.java.net/>
- Ziele:
 - API zur einfachen Erstellung von Web Services
 - Berücksichtigung von REST Prinzipien
 - Format-unabhängig (mehr als nur XML)
 - HTTP-basiert (protokollabhängig)
 - unabhängig vom Container

JAX-RS

- Klassen, Interfaces, Exceptions und Annotationen in javax.ws.rs
- Status:
 - 1.0 Final seit Oktober 2008
 - 1.1 Draft seit März 2009

Jersey

- Referenzimplementation von JAX-RS
 - JAX-RS 1.0: Jersey 1.0 (aktuell 1.0.3.1)
 - JAX-RS 1.1: Jersey 1.1 (aktuell 1.1.2 ea)
- Teil von Glassfish unter der Rubrik "Next Generation Web"
 - v3 Preview: Jersey 1.1
 - v2 und v3 Prelude: Jersey 1.0 über's Update Center

Jersey

- über JAX-RS hinaus:
 - verschiedene Deployment Möglichkeiten
 - Client API

JAX-RS Entwicklungsprozess

- Eine Klasse pro Ressource
 - Standard Java Klasse (POJO)
- Methoden zur Behandlung der HTTP Requests
 - Bereitstellung vorhandener Ressourcen
 - Speichern neuer Ressourcen
 - Operationen mit/auf Ressourcen

JAX-RS Entwicklungsprozess

- Annotationen zur Spezifizierung von
 - URI Mappings
 - Abbilden der HTTP Methoden
 - Abbilden der URI Komponenten
 - Abbilden der HTTP Header
 - Request Parameter und Rückgabe Typen
- Bei Bedarf: Erstellen von Entity Providern

Beispiel - eine Serviceklasse

```
@Path("/echoservice")
public class MyEchoService {

    String message = "";

    @GET
    @Produces("text/plain")
    public String getEchoMessage() {
        return message;
    }

    @POST
    @Consumes("text/plain")
    public void storeEchoMessage(String message) {
        this.message = message;
    }
}
```

Annotationen

- **@Path**: Identifiziert den URI Pfad
 - anwendbar auf Klassen und Methoden
 - Value setzt Pfad voraus oder definiert Variable
- **@PathParam**: Zugriff auf eine Pfadkomponente
 - anwendbar in Methodenparametern

```
@Path("/kunden/{kundenr}")  
class KundenService {  
    @GET  
    public String getMethod(@PathParam("kundenr") String nr) { ...
```

Annotationen

- **@GET, @POST, @PUT, @DELETE, @HEAD**
 - Entsprechend der HTTP Methoden
 - anwendbar auf Methoden
- **@QueryParam**
 - Erlaubt Zugriff auf HTTP Query Parameter
 - anwendbar auf Methodenparameter
- **@FormParam**
 - Zugriff auf Post Form Parameter
 - anwendbar auf Methodenparameter

Annotationen

- **@Consumes**
 - bestimmt akzeptierte Datentypen
- **@Produces**
 - bestimmt verfügbare Datentypen

Entity Provider

- Mappings zwischen Repräsentationen und Java Typen
- MessageBodyReader
 - bilden HTTP Request Body auf Methoden Parameter ab
- MessageBodyWriter
 - generieren HTTP Response Body aus Methodenwert

Entity Provider

- Default Typen
 - byte[], java.io.InputStream, java.io.Reader, java.io.File, javax.activation.DataSource (*/*)
 - java.lang.String (text/*)
 - javax.xml.transform.Source (text/xml, application/*xml)
 - MultivaluedMap<String, String> (application/x-www-form-urlencoded)
 - StreamingOutput (*/* - nur MessageBodyWriter)
 - javax.xml.bind.JAXBElement mit JAXB Klassen (text/xml, application/*xml, application/json)
- eigene Provider mit @Provider:
- @Provider
class MyProvider implements MessageBodyWriter<MyType> {...

Response und ResponseBuilder

- sollten zusätzliche HTTP Metadaten notwendig sein
 - Header
 - Status

- ResponseBuilder zum "Einpacken" der Entities

- Beispiel:

```
return Response.status(404).entity("Kenn ich nich\n")  
    .type("text/plain").build();
```

URIBuilder

- Hilft beim Erstellen von URIs (zum Verweisen auf andere Ressourcen)
- Beispiel:

```
URI uri = URIBuilder.fromUri(BASEURI)  
.path(Kunden.class)  
.path(id).build();
```

Jersey Deployment Varianten

- Plain Java SE Umgebung mittels integriertem Webserver (Grizzly)
- Servlet Umgebung durch Jersey Servlet
(`com.sun.jersey.spi.container.servlet.ServletContainer`)
- Java EE 6 Umgebung

JAX-RS Implementationen

- Jersey: <http://jersey.dev.java.net>
- Restlet mit JAX-RS Extension: <http://www.restlet.org>
- RESTEasy: <http://www.jboss.org/resteasy>
- Apache CXF: <http://cxf.apache.org>

Fragen