Aktuelle Technologien zur Entwicklung verteilter Anwendungen

RESTful Web Services mit JAX-RS

Überblick, Grundlagen und Entwicklung mit Java

Gliederung

A. RESTful Web Services mit JAX-RS

I. Web Services
II. RESTful Web Services
III. Java API for RESTful Web Services
IV. Beispielapplikation
V. Fazit

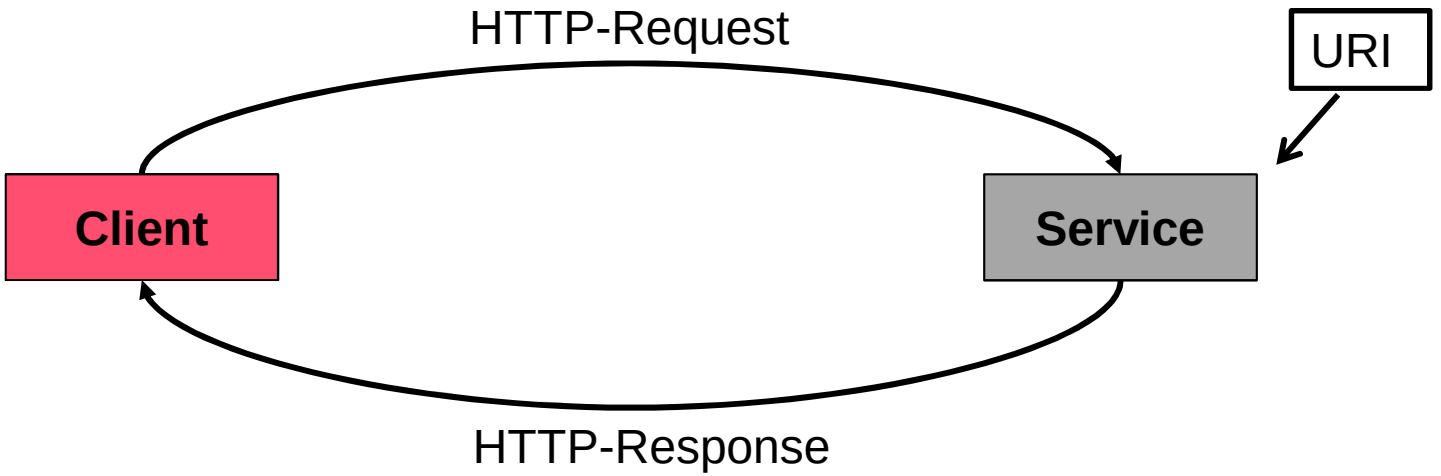
Ausgangslage

- Wachsende heterogene IT-Landschaften
- Zunehmende Komplexität von Geschäftsprozessen
- Suche nach einfacher und sicherer Kommunikation

Quelle: http://www.altvater.ch/wp-content/uploads/2012/06/9003-CRM-IT-Landschaft-001.jpg

Was ist ein Web Service?

- Client-Server Applikation
- Quasi-Standard in der SOA Welt
- Gewährleisten hohe Interoperabilität



RESTful Web Services

REST… Was ist das eigentlich?

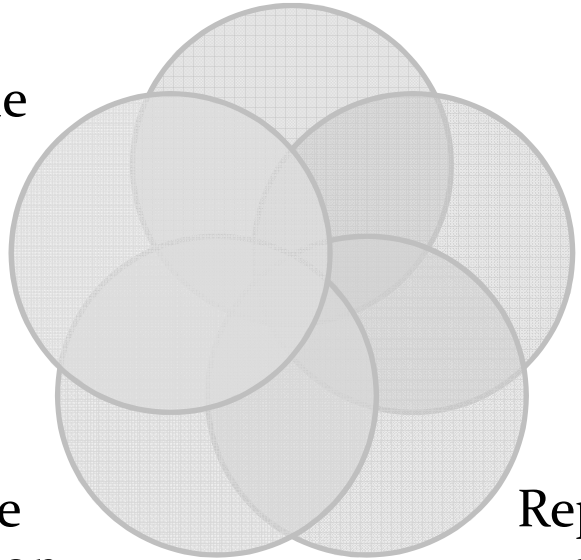
Einführung & Motivation

- **RE**presentational **S**tate **T**ransfer
- Architekturstil
- Einfache Implementierung
- Basiert auf den Prinzipien des World Wide Webs
- Lose Kopplung (Repräsentationsformat)

RESTful Architekturprinzipien

Addressierbarkeit

Hypermedia **A**s **T**he **E**ngine **O**f **A**pplication **S**tate



Einheitliche Schnittstellen

Zustandslose Kommunikation

Repräsentations-orientierung

RESTful Architekturprinzipien

```
<employees>
   <link rel="prev"href="http://www.mycompany.com/employees?startIndex=1"/>
   <link rel="next" href="http://www.mycompany.com/employees?startIndex=5"/>
   <employee id="12345">
       <name>Albert Einstein</name>
       <salary>1337 EUR</salary>
   </employee>
   ...
</employees>
```

Java API for RESTful Web Services (JAX-RS)

Web Service API

Java API for RESTful Web Services

Exkurs – HTTP (Hypertext Transfer Protocol)

- Protokoll zur Übertragung von Daten über ein Netzwerk
- HTTP Methoden:
  - @GET
  - @POST
  - @PUT
  - @DELETE
  - @HEAD
  - @OPTIONS
- Status Codes teilen den Status seiner Anfrage mit

Java API for RESTful Web Services

Exkurs – MIME-Typen

- Fand ursprünglich in der Übertragung von Emails Verwendung
- Client definiert Format der Ressource über Feld „Content-Type“ im HTTP-Header
- In JAX-RS erreichbar über Konstanten der Klasse *javax.ws.rs.core.MediaType*

Annotationsbasierter Ansatz

```
// imports
…
@Path("/employee")
public class EmployeeResource {

        @GET
        @Path("/{id}")
        @Consumes(MediaType.APPLICATION_XML)
        @Produces("application/xml")
        public Employee getEmployee(@PathParam("id") String id) {
        //TODO get employee with given id from database
        return employee;
        }
        …
}
```

Binden von URIs mit @Path

```
…
@Path("/employee")
public class EmployeeResource {
…
```

- Deklaration des Web Services mit @javax.ws.rs.Path(String path)
- Auch Methoden werden annotiert und adressiert
- String als Teil des URIs zum Web Service
- Parameter Injection möglich

Binden von HTTP Methoden

```
// imports
…
@Path("/employee")
public class EmployeeResource {

      @GET
      @Path("/{id}")
      @Consumes(MediaType.APPLICATION_XML)
      @Produces("application/xml")
      public Employee getEmployee(@PathParam("id") String id) {
      //TODO get employee with given id from database
      return employee;
      }
      …
}
```

Content Negotiation

```
// imports
…
@Path("/employee")
public class EmployeeResource {

        @GET
        @Path("/{id}")
        @Consumes(MediaType.APPLICATION_XML)
        @Produces("application/xml")
        public Employee getEmployee(@PathParam("id") String id) {
        //TODO get employee with given id from database
        return employee;
        }
        …
}
```

Parameter Injection

- Wert der @Path-Annotation ist einfacher String
- Modifikation des Strings erlaubt Übergabe von Parametern
- Beispiele für Annotationen:
  - **@javax.ws.rs.PathParam**
  - **@javax.ws.rs.QueryParam**
  - @javax.ws.rs.MatrixParam
  - @javax.ws.rs.HeaderParam
  - @javax.ws.rs.DefaultValue
  - …

Parameter Injection

- @PathParam

```
@GET
@Path("/agedPath/{age}")
@Produces({ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON })
public List<Employee> getEmployeesOverAgeWithPathParam(
            @PathParam("age") int age) {
      return DataAccess.getEmployeesOverAge(age);
}
```

Parameter Injection

- @QueryParam

```
@GET
@Path("/agedQuery")
@Produces({ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON })
public List<Employee> getEmployeesOverAgeWithQueryParam(
            @QueryParam("age") int age) {
      return DataAccess.getEmployeesOverAge(age);
}
```

Content Handler

- Message Body Reader & Writer (Entity Provider dienen dem Ver- und Entpacken von Java Objekten)
- Entity Provider werden mit der Annotation @Provider gekennzeichnet

Unterstützte Medientypen	Java Typ
Alle Medientypen (*/*)	java.lang.String
Alle Medientypen (*/*)	Java.io.InputStream, java.io.Reader
Alle Medientypen (*/*)	Javax.activation.DataSource
XML Medientypen (text/xml, application/xml, application/*+xml)	Javax.xml.transform.Source
Alle Medientypen (*/*)	Java.io.File
Alle Medientypen (*/*)	byte[]
Form Inhalt (aplication/x-www-form-urlencoded)	javax.ws.rs.core.MultivaluedMap<String,String>
Alle Medientypen (*/*), nur für MessageBodyWriter	Javax.ws.rs.core.StreamingOutput

Tabelle 1: Bereitgestellte MessageBodyReader und MessageBodyWriter

Content Handler

- JAX-RS interpretiert alle Methodenparameter ohne Annotation als Repräsentation des Bodys
- Nur ein Methodenparameter kann Body eines HTTP Requests präsentieren

HTTP-Body

```
@PUT
@Path("{id}")
@Consumes("application/xml")
public void update(@PathParam("id") int id, Employee employee) {...}
```

Einfache Response Codes

- HTTP-Methoden liefern Status Code zurück
- Status Code für erfolgreiche Responses: 200 – 399
- Status Code für Fehlermeldungen: 400-599
- Generell:
  - Erfolgreiche Anfragen mit Body → Status Code: 200 OK
  - Erfolgreiche Anfragen ohne Body → Status Code: 204 No Content

Welchen Wert liefern Methoden mit Rückgabetypen void?

Komplexe Responses

- Erzeugen eigener, modifizierter Response Objekte möglich
- Abstrakte Klasse Response enthält drei simple Methoden
  - getEntity()
  - getStatus()
  - getMetaData()

```
…
// TODO Insert into Database
Response.ResponseBuilder builder = Response.ok("Insert into Database
successful", MediaType.TEXT_PLAIN_TYPE);

builder.status(Response.Status.ACCEPTED);
return builder.build();
}
```

Exception Handling

- Werfen eigener Exceptions mittels WebApplicationException

```
…
if(employee == null) {
      throw new WebApplicationException(Response.Status.NOT_FOUND);
}
…
```

- Exception Mapper

```
@Provider
public class EntityNotFoundMapper
     implements ExceptionMapper<EntityNotFoundException> {

 public Response toResponse(EntityNotFoundException e) {
      return Response.status(Response.Status.NOT_FOUND).build();
   }
}
```

Java API for RESTful Web Services (JAX-RS)

Client API

Client API

- Client API im Paket javax.ws.rs.client
- Instanziierung des Clients mithilfe der statischen Methode newClient() in der Klasse ClientFactory

```
Client client = ClientFactory.newClient();
```

- Ressourcenzugriff über Verkettung von Methodenaufrufen

```
Response res = client
        .target("http://www.mycompany.com/employee")
        .target("http://www.mycompany.com/employee/15")
        .queryParam(name, value)
        .request(MediaType.APPLICATION_XML).get(Employee.class);
                        .request("text/plain")
                        .header("MyHeader", "...")
                        .get();
```

Targets

- Hervorragend geeignet zur Bildung komplexer URIs
- Erweiterung des URIs durch Pfadsegmente oder Templates

```
Target base = client.target("http://www.mycompany.com/employee");
Target division = base.path("division").path("{whom}");
Response res = division.pathParam("whom", "5000").request("...").get();
```

Template Parameter

Wie lautet der URI?

http://www.mycompany.com/employee/division/5000

Invocations

- Request, das bereits vorbereitet und bereit für Ausführung ist
- Client Request Invocation Builder bietet Methoden zur Vorbereitung

```
Invocation inv; inv=client.target("http://www.mycompany.com/employee")
.request("application/xml").buildGet();

            //synchron
List<Employee> empList1 = inv.invoke(new GenericType<List<Employee>>() {});

            //asynchron
Future<List<Employee>> empList2 = inv.submit(new GenericType<List<Employee>>()
{});
try {
      List<Employee> test = empList2.get();
} catch (InterruptedException | ExecutionException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
}
```

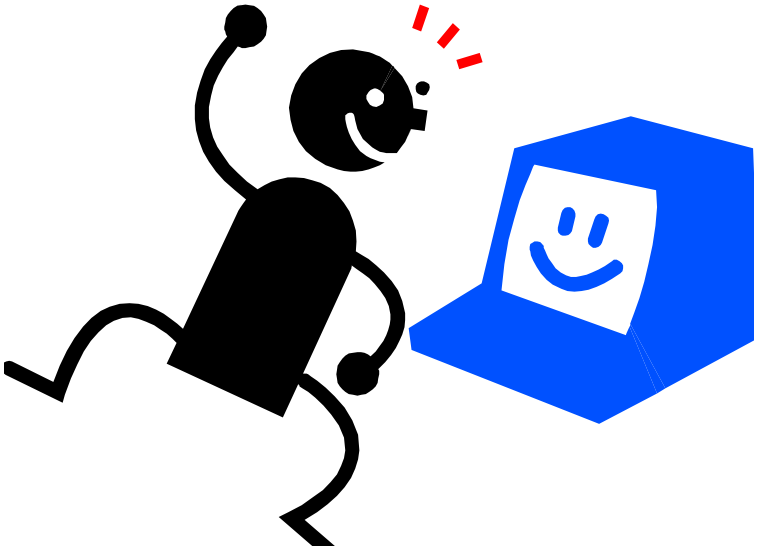
Java Architecture for XML Binding

- Framework bzw. Spezifikation, kein Standard von JAX-RS
- Bietet dem Entwickler eine einfache Möglichkeit, Java Klassen mittels Annotationen an ein XML Schema zu binden
- Annotationsbasiert
  - @XmlRootElement
  - @XmlAccessorType
  - @XmlAttribute
  - @XmlElement

Beispiel: Siehe Formate via Advanced REST Client

Beispiel: Employee-Applikation

- Die Beispielapplikation
- JAX-RS in Verbindung mit JAXB (Advanced REST Client)
- Deployment



Fazit

- REST ist keine Technologie
- REST ist schlank
- Einfache Implementierung mit JAX-RS
- Große REST Community
- Client vervollständigt die JAX-RS Spezifikation
- Eigene Erfahrungen

Vielen Dank für Ihre Aufmerksamkeit!

Noch jemand da...?

Wach...?

Fragen...?

Literaturverzeichnis

- *REST Patterns, MIME Type*. (2008). Abgerufen am 04. April 2013 von http://restpatterns.org/Glossary/MIME_Type
- *redhat: RESTEasy JAX-RS, RESTFul Web Services for Java*. (2012). Abgerufen am 13. April 2013 von http://docs.jboss.org/resteasy/docs/2.3.1.GA/userguide/html_single/index.html
- *Oracle: The Java EE 6 Tutorial, Stand 2012*. (02. April 2013). Von http://www.docs.oracle.com/javaee/6/tutorial/doc/gijvh.html abgerufen
- Burke, B. (2009). *RESTful Java with JAX-RS.* Sebastapol: O`Reilly.
- *Oracle: JavaTM Platform, Enterprise Edition 6 API Specification, Package javax.ws.rs, Stand Januar 2013*. (kein Datum). Abgerufen am 02. April 2013 von http://docs.oracle.com/javaee/6/api/javax/ws/rs/package-summary.html
- Pericas-Geertsen, S. (2013). *JAX-RS: Java API for RESTful Web Services*. Abgerufen am 29. April 2013 von http://download.oracle.com/otn-pub/jcp/jaxrs-2_0-pfd-spec/339-spec.pdf?AuthParam=1367262282_3929e3836cff9fc3fcaad2dae38cd0a8
- Snell, J. (2001). *Programming Web Services with SOAP.* Sebastopol: O`Reilly.
- *W3C: Hypertext Transfer Protocol - HTTP/1.1*. (kein Datum). Abgerufen am 04. April 2013 von http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html