JUST DO IT
HÖHERE ABTEILUNG FÜR INFORMATIONSTECHNOLOGIE

tgm
Die Schule der Technik

Laborprotokoll
SOA and RESTful Webservice

Dezentrale Systeme
5BHITT 2015/16

Paul Kalauner
Patrick Malik

Version 0.2

Note:

Betreuer: Th.Micheler

Begonnen am 04. Dezember 2015

Beendet am 17. Dezember 2015

Inhaltsverzeichnis

Einführung

Das neu eröffnete Unternehmen **iKnow Systems** ist spezialisiert auf **Knowledgemanagement** und bietet seinen Kunden die Möglichkeiten Daten und Informationen jeglicher Art in eine Wissensbasis einzupflegen und anschließend in der zentralen Wissensbasis nach Informationen zu suchen (ähnlich wikipedia).

Folgendes ist im Rahmen der Aufgabenstellung verlangt:

- Entwerfen Sie ein Datenmodell, um die Eintraege der Wissensbasis zu speichern und um ein optimitiertes Suchen von Eintraegen zu gewaehrleisten. **[2Pkt]**
- Entwickeln Sie mittels RESTful Webservices eine Schnittstelle, um die Wissensbasis zu verwalten. Es muessen folgende Operationen angeboten werden:
  - **Hinzufuegen** eines neuen Eintrags
  - **Aendern** eines bestehenden Eintrags
  - **Loeschen** eines bestehenden Eintrags

  Alle Operationen muessen ein Ergebnis der Operation zurueckliefern. **[3Pkt]**
- Entwickeln Sie in **Java** ein **SOA Webservice**, dass die Funktionalitaet **Suchen** anbietet und das **SOAP** Protokoll einbindet. Erzeugen Sie fuer dieses Webservice auch eine**WSDL**-Datei. **[3Pkt]**
- Entwerfen Sie eine **Weboberflaeche**, um die **RESTful Webservices** zu verwenden. **[3Pkt]**
- Implementieren Sie einen **einfachen Client** mit einem User Interface (auch Commandline UI moeglich), der das **SOA Webservice** aufruft. **[2Pkt]**
- Dokumentieren Sie im weiteren Verlauf den Datentransfer mit SOAP. **[1Pkt]**
- Protokoll ist erforderlich! **[2Pkt]**

Info:
Gruppengroesse: 2 Mitglieder
Punkte: 16

Zum Testen bereiten Sie eine Routine vor, um die Wissensbasis mit einer **1 Million Datensaetze** zu fuellen. Die Datensaetze sollen mindestens eine Laenge beim Suchbegriff von 10 Zeichen und bei der Beschreibung von 100 Zeichen haben! **Ist die Performance bei der Suche noch gegeben?**

Links:
JEE Webservices:
http://docs.oracle.com/javaee/6/tutorial/doc/gijti.html

Apache Web Services Project:
http://ws.apache.org/

Apache Axis/Axis2:
http://axis.apache.org
http://axis.apache.org/axis2/java/core/

IBM Article: Java Web services - JAXB and JAX-WS in Axis2:
http://www.ibm.com/developerworks/java/library/j-jws8/index.html

Ergebnisse

1.1 Datenmodell

Das Datenmodell ist aufgrund der Aufgabenstellung und des Projektumfangs nicht sonderlich komplex und besteht aus einer Tabelle mit den Attributen:

- id
  - ein long bzw. bigint
  - PK
  - autoincrement
- title
  - String bzw. VARCHAR(50)
  - NOT NULL
- content
  - String bzw. longtext/CLOB

Die Erstellung der Tabelle wurde gänzlich von Spring übernommen (intern Hibernate. Die zugehörige Klasse Entry gibt wie aus Hibernate bekannt die notwendigen Daten dazu an:

```
@Id
@GeneratedValue
private long id;

@NotEmpty
@Size(max = 50)
private String title;

@Lob
private String content;
```

Wobei @Id angibt das es sich um einen PK handelt, @GeneratedValue --> Autoincrement, @NotEmpty --> Not Null und @Lob --> CLOB/BLOB (je nachdem welcher Typ hier mapped) @Size gibt zudem die Größe an (hier VARCHAR(50))

1.2 REST

Das Management der REST-Anfragen erfolgt im Package „rest" in der Klasse „EntryController". Beispielsweise sieht das Mapping auf die URL */entries/<id>* um einen Eintrag mit der angegebenen ID abzurufen folgendermaßen aus:

```
@RequestMapping(value = "/entries/{id}", method = RequestMethod.GET)
public ResponseEntity<Entry> getById(@PathVariable("id") long id) {
    return new ResponseEntity<>(entryDao.getById(id), HttpStatus.OK);
}
```

-Zusätzliche Dokumentation folgt-

Die RESTful-Webservices werden durch eine Webapplikation aufgerufen. Diese wurde mithilfe von Spring MVC umgesetzt. Der Controller hierfür befindet sich im Package „webinterface". Unter „resources/templates" sind die HTML-Dateien zu finden. Zusätzlich wurde Bootstrap verwendet.

Die Restservices werden von Spring aus mittels den sogenannten „RestTemplates" aufgerufen. Solch ein Aufruf sieht folgendermaßen aus:

```
Entry result = restTemplate.postForObject("http://localhost:8080/entries"
, entry, Entry.class);
```

Natürlich könnte auch aus dem Webcontroller direkt auf die Datenbank zugegriffen werden. Allerdings war der Sinn der Übung, die RESTful-Webservices zu nutzen.

1.3 SOA

WSDL/XSD

Für den Webservice musste ein WSDL-File (Web Service Definition Language) generiert werden. Dieses hängt mit einer .xsd Datei zusammen, welche die Eigenschaften des Webservices bereits beschreibt.

Hier werden verschiedene Elemente (in diesem Fall die Methoden) und Typen (bspw. das zugehörige Objekt) definiert.

```
<xs:element name="getDataResponse">
     <xs:complexType>
          <xs:sequence>
               <xs:element maxOccurs="unbounded" minOccurs="0"
          nillable="true" name="entry" type="tns:entry"/>
          </xs:sequence>
     </xs:complexType>
</xs:element>
```

Hier wird die Funktion getDataResponse definiert, welche eine Collection (maxOccurs="unbounded") vom Typen entry (type="tns:entry") zurückgibt. "tns" steht hier für Target Namespace. Der Type entry wird im selben xsd File definiert:

```
<xs:complexType name="entry">
     <xs:sequence>
          <xs:element name="content" type="xs:string"
     minOccurs="0"/>
          <xs:element name="id" type="xs:long"/>
          <xs:element name="title" type="xs:string" minOccurs="0"/>
     </xs:sequence>
</xs:complexType>
```

Wobei "complexType" besagt das es sich um einen komplexen Typen handelt -> Objekt o.ä..
Der komplexe Typ besteht aus 3 Elementen wovon 2 Strings sind und die id als long definiert wurde.

Zudem wurde in diesem File noch der "getDataRequest" definiert, ähnlich wie die erste.

Jaxb bietet einem die Möglichkeit aus dem xsd-File die Klassen zu definieren. [1]

Webservice

Die nun generierten Klassen müssen mit dem tatsächlichen Webservice verbunden werden. Spring bietet große Unterstützung bei der Anbindung der

Klassen und SOAP ansich. [2]

Repository

Ein sog. Repository stellt die Daten für den Dienst zur Verfügung, da die Datenbankanbindung bereits implementiert ist, kann diese hier einfach verwendet werden und somit Daten aus der DB zur Verfügung stellen. Die Annotation @Repository ist hier notwendig, ein Hinweis (an Spring), dass es sich um eine Klasse handelt, welche Daten zur Verfügung stellt.

Endpoint

Der Endpoint, auch mit @Endpoint gekennzeichnet, ist für die tatsächliche Anbindung zuständig. Im Endpoint wird definiert durch welchen Request welche Response zurückgegeben wird, bzw. womit diese Response befüllt wird. Die Responsemethode muss mit @ResponsePayload versehen werden, damit der zurückgegebene Wert auf den Rückgabewert der Response gemapped wird. Als Parameter muss dieser Methode der Request übergeben werden (zuvor mit JAXB generiert) und dieser muss mit @RequestPayload gekennzeichnet werden, um zu gewährleisten, dass die Requests auf diesen Parameter bzw. diese Methode gemapped werden.

Web Service Beans

In dieser Klasse werden die notwendigen Beans konfiguriert. Unter Anderem wo das generierte wsdl File abgelegt wird, welche .xsd Datei verwendet werden soll, und andere Spezifikationen für das wsdl File werden getroffen.

Client

Der Client ist von Spring unabhängig, es wurde javax.xml.soap für den gesamten Clientteil verwendet. Dem Client wird zu Beginn die Adresse des SOAP-Servers und der Port angegeben, danach befindet sich der User im Programm und kann bis zur Eingabe von "\q" die Suche verwenden. Es werden nur Ergebnisse angezeigt die dem eingegebenen Namen exakt entsprechen.

Die Argumente wurden mit Apache Commons CLI[3] geparsed und im Fehlerfall mit Log4J[4] gelogged.

Der Response wird vom Envelope zurück auf einen Entry gemapped und dann schön formatiert zurückgegeben.

1.4 Probleme

Der Client macht insofern Probleme als dass man ihn alleine für unsere Konfiguration verwenden kann. SOAP bietet einem leider nicht die Möglichkeit den Client dynamisch zu gestalten.

Es ist nicht möglich in der xsd auf bereits bestehende Klassen als Datentyp zuzugreifen. Entweder müssen die generierten Klassen umgeschrieben werden oder es muss ein Mapping stattfinden. Wir haben uns für das Mapping entschieden.

Ausführung der Anwendung

Zur Ausführung wird eine MySQL-Datenbank benötigt.
Zuerst sollte in der Server-Jar die IP-Adresse der Datenbank gesetzt werden. Hierfür öffnet man das JAR-File mit 7Zip, WinRAR oder ähnlichen Programmen. In der Datei „applications.properties" unter dem Punkt „spring.datasource.url" kann nun der Hostname der Datenbank angegeben werden. Nach dem Hostname folgt ein / und danach der Name der Datenbank, welche bereits existieren muss.
Bei spring.datasource.username und spring.datasource.password muss der Username und das Passwort des Users festgelegt werden.

Mittels folgendem Kommando kann der Server gestartet werden:

```
java –jar Kalauner_Malik_Dezsys07_Server.jar <Anzahl der einzufügenden Testdaten>
```

Falls kein Argument angegeben ist, werden keine neuen Testdaten hinzugefügt.

Das Webinterface ist nun unter http://localhost:8080 erreichbar.

Der Client kann ebenfalls mit `java –jar` gestartet werden. Er hat folgende Argumente:

-a,--address <arg>	The hostaddress of the soap server
-h,--help	help
-p,--port <arg>	The port of the soap server

Quellen

[1] Project JAXB, java.net, 03.02.2015, https://jaxb.java.net/ , 17.12.2015

[2] Producing a SOAP Webservice, Pivotal Software, 17.12.2015, https://spring.io/guides/gs/producing-web-service/ , 17.12.2015

[3] Commons CLI, Apache Commons, 17.06.2015, https://commons.apache.org/proper/commons-cli/ , 17.12.2015

[4] Apache Log4J, Apache Logging, 10.12.2015, http://logging.apache.org/log4j/2.x/ ,17.12.2015