Read-Write Resources JAX-RS & JAXB

Laboratory of Service Design and Engineering

2011/2012

Outline

- Designing Read/Write Resource-Oriented Service
- JAX-RS + JAXB
- Examples
- Exercise

Designing Read/Write Resource-Oriented Services

How to create a RESTful Web Service?

1. Figure out the data set
2. Split the data set into resources
   - For each kind of resource:
3. Name the resources with URIs
4. Expose a subset of the uniform interface
5. Design the representation(s)
6. Link the resources to each other
7. What's supposed to happen?
8. What might go wrong?

DAO: Data Access Object

- A DAO is an object that provides an abstract interface to some type of persistence mechanism, providing some specific operations without exposing details of the database

DAO: Data Access Object

- Many possible implementations, depending on the way of managing the persistence
- JDBC
  - http://java.sun.com/docs/books/tutorial/jdbc/
- Hibernate
  - https://www.hibernate.org/
- Db4o
  - http://www.db4o.com/about/productinformation/resources/db4o-7.4-tutorial-java.pdf

Exercise

- Examine your data model (Student data model)
- Divide your project with 3 layers
  - Data Layer
  - Business Logic Layer
  - Presentation/ Service Layer
- Make your entities from data model with JAXB annotations
- Implement business logic layer
  - Get data from service to marshall
  - Provide data to service (unmarshall)

Exercise

- Implement service layer
  - Think about resources
  - Provide uniform interface
  - Provide CRUD operation through services
- Use REST client to interact with services
- Presentation of data should be in XML format

Person class

```
@XmlAccessorType(value = XmlAccessType.FIELD)
public class Person {
    @XmlElement
    private String name;
    @XmlElement
    private int age;
    @XmlElement
    private String address;

    public Person() {
    }

    public Person(String name, int age, String address) {
        this.name = name;
        this.age = age;
        this.address = address;
    }

    /**
     * @return the name
     */

    public String getName() {
        return name;
    }
```

Student Class

```
@XmlAccessorType(value = XmlAccessType.FIELD)
@XmlRootElement
public class Student extends Person{


    @XmlElement
    private int registrationNo;
    @XmlElement
    private String course;
    @XmlElement
    private int marks;

    public Student() {
    }

    public Student(int regno,String course, int marks,Person person) {
        super(person.getName(),person.getAge(), person.getAddress());
        this.registrationNo=regno;
        this.course = course;
        this.marks = marks;
```

StudentList

```
@XmlAccessorType(value = XmlAccessType.FIELD)
@XmlRootElement(name ="Students")
public class StudentList {

    @XmlElements({@XmlElement(name="Student",type=Student.class)})
    private ArrayList<Student> studentList;

    public StudentList() {
        studentList = new ArrayList<Student>();
    }

    /**
     * @return the studentList
     */
    public ArrayList<Student> getStudentList() {
        return studentList;
    }

    /**
     * @param studentList the studentList to set
     */
    public void setStudentList(ArrayList<Student> studentList) {
        this.studentList = studentList;
    }
```

Marshalling & Unmarshalling

```
public void doMarshalling(StudentList students)
{
    try {
        JAXBContext context = JAXBContext.newInstance(StudentList.class);
        Marshaller marshaller = context.createMarshaller();
        marshaller.setProperty(marshaller.JAXB_FORMATTED_OUTPUT, true);
        marshaller.marshal(students, new FileOutputStream(new File("f://person.xml")));

    } catch (FileNotFoundException ex) {
        Logger.getLogger(JavaMarshaller.class.getName()).log(Level.SEVERE, null, ex);
    } catch (JAXBException ex) {
        Logger.getLogger(JavaMarshaller.class.getName()).log(Level.SEVERE, null, ex);
    }
```

```
public StudentList doUnMarshalling()
{
    StudentList students=null;
    try {
        JAXBContext context = JAXBContext.newInstance(new Class[]{org.soaa.entities.StudentList.class});
        Unmarshaller unmarshaller = context.createUnmarshaller();
        students =  (StudentList)unmarshaller.unmarshal(new File("f://person.xml"));
    } catch (JAXBException ex) {
        Logger.getLogger(XMLUnMarshaller.class.getName()).log(Level.SEVERE, null, ex);
    }

    return students;
}
```

StudentResource (Services)

```
@Path("/student")
@GET
@Produces("application/xml")
public Response getStudent() {

    XMLUnMarshaller unmarshaller = new XMLUnMarshaller();
    StudentList students = unmarshaller.doUnMarshalling();
    return Response.ok(students).build();

}

@Path("/student/{name}")
@GET
@Produces("application/xml")
public Response getStudentByName(@PathParam("name") String name) {

    Student st = new Student();
    XMLUnMarshaller unmarshaller = new XMLUnMarshaller();
    StudentList students = unmarshaller.doUnMarshalling();
    for (Student student : students.getStudentList()) {
        System.out.println(student.getName());
        if (student.getName().equals(name)) {
            st = student;
            break;
        }
```

Assignment 2 (deadline: 16-Nov)

- Refine your existing data model of bibliography.
- Split the data set into resources (e.g., Researcher, Publication, Department etc)
- Provide create, retrieve, update and delete services on top of major resources.
  - For example, a retrieve resource should provide (e.g. get all, get by ID, get by Name etc)
- Persist data (using DAO) into some database (Oracle, MySQL)
- For the client you may use any REST client
- Your presentation must be in XML and JSON.