Web Applications Developments

ITWT413

LECTURE 7: JAKARTA EE CORE PROFILE

DR. HALA SHAARI

Course Structure

- Lecture (Monday& Tuesday)
- Time: 12:30-14:00
- Microsoft Teams Code (goi2d67)

(Lectures, labs, announcements, References):

- Grading :
  - 25% Midterm exam.
  - 25% Assignments
  - 25% Group Project (Groups of two).
  - 25% Final Exam.

Course References

Our Course Main References:

- Jakarta EE Tutorial
- Java EE to Jakarta EE 10 Recipes: A Problem-Solution Approach for Enterprise Java (2022)

key course objectives

1) Master Core Jakarta EE Concepts: Understand the fundamental architecture and components of Jakarta EE, including Servlets, JSP, JSF, and Enterprise JavaBeans (EJB).
2) Develop RESTful Web Services: Build and deploy scalable RESTful APIs using Jakarta RESTful Web Services (JAX-RS), with advanced features like exception handling, filters, and security.
3) Implement Dependency Injection and Persistence: Leverage Contexts and Dependency Injection (CDI) and Jakarta Persistence (JPA) to manage beans and database interactions in enterprise applications.
4) Ensure Application Security and Transaction Management: Apply Jakarta EE's security framework for authentication, authorization, and manage transactions using declarative and programmatic approaches.
5) Deploy Cloud-Native Applications: Utilize Jakarta EE to build, containerize, and deploy microservice-based applications in cloud environments, incorporating tools like Docker and Kubernetes.

Jakarta EE Tutorial- Structure

Basic Platform	Jakarta EE Core Profile	Jakarta EE Web Profile	Jakarta EE Platform
Resource Creation	Jakarta CDI Lite	Jakarta CDI Full	Jakarta Mail
Injection	Jakarta REST	Jakarta Validation	Jakarta Messaging
Packaging	Jakarta JSON	Jakarta Security	Jakarta Batch
		Jakarta Servlets	
		Jakarta Faces	
		Jakarta WebSocket	
		Jakarta Persistence	
		Jakarta Enterprise Beans Lite	

Lecture Agenda

- JSON Binding in the Jakarta EE Platform
  - Overview of the JSON Binding API
    - Creating a jasonb Instance
    - Using the Default Mapping
    - Using Customizations
    - Using Annotations
  - Running the jsonbbasics Example Application
  - Components of the jsonbbasics Example Application
  - Running the jsonbbasics Example Application

Jakarta JSON

- JSON is a data exchange format widely used in web services and other connected applications.
- The Jakarta JSON Binding specification provides a standard binding layer (metadata and runtime) between Java classes and JSON documents.
- One Jakarta JSON Binding reference implementation is Yasson, which is developed through Eclipse.org and is included as part of GlassFish Server.

JSON Binding in the Jakarta EE Platform

- Jakarta EE includes support for the Jakarta JSON Binding spec, which provides an API that can serialize Java objects to JSON documents and deserialize JSON documents to Java objects.
- Jakarta JSON Binding contains the following packages:
  - The jakarta.json.bind package
  - The jakarta.json.bind.adapter
  - The jakarta.json.bind.annotation package
  - The jakarta.json.bind.config package
  - The jakarta.json.bind.serializer package
  - The jakarta.json.bind.spi package

Main Classes and Interfaces in jakarta.json.bind

Class or Interface	Description
`Jsonb`	Contains the JSON binding methods for serializing Java objects to JSON and de-serailizing JSON to Java objects.
`JsonBuilder`	Used by clients to create `Jsonb` instances.
`JsonbConfig`	Used to set configuration properties on `Jsonb` instances. Properties include binding strategies and properties for configuring custom serializers and deserializers.
`JsonbException`	Indicates that a problem occurred during JSON binding.

Main Classes and Interfaces in jakarta.json.bind.config

Class or Interface	Description
`PropertyNamingStrategy`	Used to set how property names are translated.
`PropertyVisibilityStrategy`	Used to set whether fields and methods should be considered properties overriding the default scope and field access behavior.
`BinaryDataStrategy`	Used to set binary encoding.
`PropertyOrderStrategy`	Used to set how properties are ordered during serialization.

Main Classes and Interfaces in jakarta.json.bind.serializer

Class or Interface	Description
`JsonbDeserializer`	Used to create a deserialization routine for a custom type.
`JsonbSerializer`	Used to create a serialization routine for a custom type.

What is Serialization in Java?

- Serialization in Java is the concept of representing an object's state as a byte stream.
- The byte stream has all the information about the object.
- Usually used in Hibernate, JMS, JPA, and EJB, serialization in Java helps transport the code from one JVM to another and then de-serialize it there.

- Deserialization is the exact opposite process of serialization where the byte data type stream is converted back to an object in the memory.
- The best part about these mechanisms is that both are JVM-independent, meaning you serialize on one JVM and de-serialize on another.

What are the Advantages of Serialization?

- Used for marshaling (traveling the state of an object on the network)
- To persist or save an object's state
- JVM independent
- Easy to understand and customize

Points to Note About Serialization in Java?

- Serialization is a marker interface with no method or data member
- You can serialize an object only by implementing the serializable interface
- All the fields of a class must be serializable
- The child class doesn’t have to implement the Serializable interface, if the parent class does
- The serialization process only saves non-static data members, but not static or transient data members
- By default, the String and all wrapper classes implement the Serializable interface

Example for Serialization in Java

```java
import java.io.*;
public class Student implements java.io.Serializable{
    public String stu_Name;
    public String stu_Addr;
    public int stu_Id;
    public static void main(String [] args){
        Student s = new Student();
        s.stu_Name = "George";
        s.stu_Addr = "ABC, XYZ";
        s.stu_Id = 1;
        try{
            FileOutputStream fileOut =
            new FileOutputStream("s.txt");
            ObjectOutputStream out = new ObjectOutputStream(fileOut);
            out.writeObject(s);
            out.close();
            fileOut.close();
            System.out.printf("Object serialized and saved in s.txt");
        } catch(IOException i){
            i.printStackTrace();
        }
    }
}
```

Result

CPU Time: 0.22 sec(s), Memory: 33516 kilobyte(s) compiled and executed in 0.812 sec(s)

```
Object serialized and saved in s.txt
```

Overview of the JSON Binding API

- This section provides basic instructions for using the Jakarta JSON Binding client API.
- The instructions provide a basis for understanding the Running the jsonbbasics Example Application.
  - Creating a jasonb Instance
  - Using the Default Mapping
  - Using Customizations
  - Using Annotations

Creating a jasonb Instance

- A jsonb instance provides access to methods for binding objects to JSON.
- A single jsonb instance is required for most applications.
- A jsonb instance is created using the JsonbBuilder interface, which is a client's entry point to the JSON Binding API. For example:

```
Jsonb jsonb = JsonbBuilder.create();
```

Using the Default Mapping

- Jakarta JSON Binding provides default mappings for serializing and deserializing basic Java and Java SE types as well Java date and time classes.
- To use the default mappings and mapping behavior, create a josnb instance and use the toJson method to serialize to JSON and the fromJson method to deserialize back to an object.
- The following example binds a simple Person object that contains a single name field

```
Jsonb jsonb = JsonbBuilder.create();

Person person = new Person();
person.name = "Fred";

Jsonb jsonb = JsonbBuilder.create();

// serialize to JSON
String result = jsonb.toJson(person);

// deserialize from JSON
person = jsonb.fromJson("{name:\"joe\"}", Person.class);
```

Using Customizations

- Jakarta JSON Binding supports many ways to customize the default mapping behavior.
- For runtime customizations, a JsonbConfig configuration object is used when creating the jsonbinstance.
- The JsonbConfig class supports many configuration options and also includes advanced options for binding custom types.
- For advanced options, see the JsonbAdapter interface and the JsonbSerializer and JsonbDeserializer interfaces.
- The following example creates a configuration object that sets the FORMATTING property to specify whether or not the serialized JSON data is formatted with linefeeds and indentation.

```
JsonbConfig config = new JsonbConfig()
    .withFormatting(true);

Jsonb jsonb = JsonbBuilder.create(config);
```

Using Annotations

- Jakarta JSON Binding includes many annotations that can be used at compile time to customize the default mapping behavior.
- The following example uses the @JsonbProperty annotation to change the name field to person-name when the object is serialized to JSON.

```
public class Person {
    @JsonbProperty("person-name")
    private String name;
}
```

The resulting JSON document is written as:

```
{
    "person-name": "Fred",
}
```

Running the jsonbbasics Example Application

- This section describes how to build and run the jsonbbasics example application. This example is a web application that demonstrates how to serialize an object to JSON and how to deserialize JSON to an object.
- The jsonbbasics example application is in the jakartaeeexamples/tutorial/web/jsonb/jsonbbasics directory.

Components of the jsonbbasics Example Application

The jsonbbasics example application contains the following files.

1. Two Jakarta Faces pages.
   - The index.xhtml page contains a form to collect data that is used to create a Person object.
   - The jsongenerated.xhtml page contains a text area that displays the data in JSON format.
2. The JsonbBean.java managed bean, which is a session-scoped managed bean that stores the data from the form and directs the navigation between the Facelets pages. This file contains code that uses the JSON Binding API.

Running the jsonbbasics Example Application

- This section describes how to run the jsonbbasics example application from the command line using Maven.
- To run the jsonbbasics example application using Maven:
  - Make sure that GlassFish Server has been started (see Starting and Stopping GlassFish Server).
  - In a terminal window, go to: jakartaee-examples/tutorial/web/jsonb/jsonbbasics
- Enter the following command to deploy the application:
  - mvn install
- Open a web browser window and enter the following address:
  - http://localhost:8080/jsonbbasics/
- Enter data on form and click Serialize to JSON to submit the form. The following page shows the JSON format of the object data.
- Click Deserialize JSON. The index page displays and contains the fields populated from the object data.

Questions?