

Web Servisleri ve Hibernate

Burak Bayramlı

Bu belge, \LaTeX ile üretilmiştir

Hibernate

- EJB3'te veri tabanına erişim `javax.persistence` paketi üzerinden yapılmaktadır. Bu paketin çıkış noktası Hibernate isimli kalıcılık (persistence) aracıdır.
- Veriye erişim birimi artık JDBC'de olduğu gibi tablolar, alanlar değil, Java kodlamasından bize tanıdık olan Java bean'lerdir.

POJO

```
public class Musteri {  
  
    String isim;  
    String soyad;  
    String ePosta;  
  
    public String getIsim() {  
        return this.isim;  
    }  
}
```

```
public String getIsim() {  
    return this.isim;  
}  
public void setSoyad(String soyad) {  
    this.soyad = arg.soyad;  
}  
public String getSoyad() {  
    return this.soyad;  
}  
}
```

POJO

```
@Entity
public class Musteri {
    @Id
    String isim;
    String soyad;
    @Column(name="e_posta")
    String ePosta;

    public String getIsim() {
        return this.isim;
    }
}
```

```
}  
public String getIsim() {  
    return this.isim;  
}  
public void setSoyad(String soyad) {  
    this.soyad = arg.soyad;  
}  
public String getSoyad() {  
    return this.soyad;  
}  
}
```

Kullanım

- Hibernate kullanmak isteyen EJB bileşenleri `@PersistenceContext` annotation'ı kullanarak bir `EntityManager` enjekte edebilir.

```
@PersistenceContext  
EntityManager em;
```

- Artık EJB, `em` referansı kullanılarak `EntityManager` metotlarına erişilebilecektir. `ID` kullanarak obje

yüklemek için `find`, bir nesneyi veri tabanına yazmak için `persist` bu çağrılara bir örnektir.

- Meselâ ID'si bilinen bir nesneyi yüklemek için

```
Musteri musteri =  
    (Musteri)em.find(Musteri.class, 12345)
```

- Yeni bir nesneyi veri tabanına yazmak için

```
Musteri musteri = new Musteri();  
musteri.setIsim(..);  
musteri.setSoyad(..);  
em.persist(musteri);
```

PersistenceContext

Bu annotation, Hibernate ayarlarını yükleyip, erişimi hazırlamak ile sorumludur. Ayarlar EJB3 standartına göre `persistence.xml` adlı bir dosyadadır.

persistence.xml

```
<persistence>
  <persistence-unit name="kitapdemoDatabase">
    <provider>org.hibernate.ejb.HibernatePersistence</provider>
    <jta-data-source>java:[DATA SOURCE ISMI]</jta-data-source>
    <properties>
      <property name="hibernate.hbm2ddl.auto" value="create-drop"/>
      <property name="hibernate.show_sql" value="true"/>
      <property name="dialect">org.hibernate.dialect.MySQLDialect</property>
      <property name="hibernate.transaction.manager_lookup_class"
        value="org.hibernate.transaction.JBossTransactionManagerLookup"/>

    </properties>
  </persistence-unit>
</persistence>
```

persistence.xml

- `create-drop`: `@Entity` ile işaretlenmiş tüm class'ların tablosu otomatik olarak veri tabanında yaratılsın (geliştirme için mükemmel bir araç).
- `show_sql`: Hibernate işlemlerinin ürettiği SQL ekrana basılsın mı?
- `dialect`: Hangi veri tabanına bağlanalım?

Data Source

- `persistence.xml` veri tabanı bağlantısı işini bir data source'a bırakabilir.
- Bu tanımları kendi de yapabilirdi, fakat buna dış bir kavrama bırakınca, production ayarlaması daha rahat oluyor.
- Bağlantı havuzu (connection pooling) tanımı data source içinde yapılabilir.

- Bir data source tanımlamak için sonu `-ds.xml` ile biten bir dosya yeterli. Bunun örneğini `resources` → `/kitapdemo-ds.xml`'de görebilirsiniz.
- `persistence.xml`'in bu DS'e sadece ismiyle referans etmesi yeterlidir.
- Data source içindeki `connection-url` JDBC'ye verilecek bağlantının ayarıdır: `jdbc:mysql://` → `localhost:3306/[DB ISMI]` şeklinde olması gerekir.

import.sql

- EJB3'ün bir avantajı daha, geliştirme amaçlı olarak CLASSPATH'te bulunduğu import.sql adlı dosyayı şemayı yarattıktan sonra DB üzerinde işletmesidir.
- Bu dosya örnek veri yüklemek için mükemmel bir araç
- Bizim master script'teki ear yaratan target'imiz, resources altında bulunduğu import.sql'i otomatik olarak ear içine koyacaktır.

- `import.sql`'de tipik olarak bol bol `INSERT` ibareleri olur. Boş ise hiç veri yüklenmez.

Web Servis

```
@Stateless
@WebService ...
public class MyService
{
    @PersistenceContext
    EntityManager em;

    @TransactionAttribute
    (TransactionAttributeType.REQUIRED)
    @WebMethod
```

```
public X getX(..) {  
    em...  
}  
..  
}
```

Görevler

- Car adlı bir kalıcı class yaratın. Üzerinde `licensePlate` ve `description` adlı iki öge olsun. Nesnenin ID'si `licensePlate` ögesi olacak.
- SQLyog'u kullanarak MySQL'inizde `cars` adlı bir taban yaratın.
- Data source'unuzdaki ayarları yaparak veri tabanı bağlantısını sağlayın.

- `resources/META-INF` altına `persistence.xml` dosyanızı koyun, derleme sistemi otomatik olarak bu dosyayı alıp işleme koyacaktır. Bu dosyanın data source bağlantısını kurun.
- `import.sql` içinde birkaç örnek Car verisi eklenmesini sağlayın. Insert formatı: `INSERT INTO TABLO (... , ...) VALUES (... , ...)`.
- `CarService` adlı bir Web Servisi yazın. Bu servisin `getCar` adlı bir metodu olsun, bu metod parametre geçilen `licensePlate`'e göre veri tabanında bu ID'ye tekabül eden Car objesini yükleyerek döndürsün. Ayrıca

`getAllCars` adlı bir metot daha olsun, bu DB'deki tüm arabaları bir liste olarak döndürecek. Bu listenin çağıran tarafta ekrana basılmasını sağlayın.

```
List<Musteri> list =  
    em.createQuery("from Musteri").getResultList();
```

Not: Bir `List<Musteri>` listesini `Musteri[]`'ye çevirmek için `List` üzerindeki `toArray` metodu kullanılabilir:

```
List<Musteri> list = ...  
Musteri[] tmpList = new Musteri[0];  
tmpList = list.toArray(tmpList);
```

Import

```
import javax.ejb.Stateless;  
import javax.ejb.TransactionAttribute;  
import javax.ejb.TransactionAttributeType;  
import javax.persistence.PersistenceContext;  
import javax.persistence.EntityManager;  
import org.bilgidata.kitapdemo.pojo.*;  
import java.util.List;  
import javax.jws.WebService;  
import javax.jws.WebMethod;  
import javax.jws.soap.SOAPBinding;
```

```
import javax.xml.soap.SOAPBinding.Style;
```

Import

```
import javax.persistence.Entity;
import javax.persistence.Column;
import javax.persistence.Id;
import javax.persistence.ManyToOne;
import javax.persistence.JoinColumn;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlRootElement;
```

```
import java.io.Serializable;
```