Association Mapping

This chapter will look at the mapping of association types of two classes, and examine mapping method of various Collection and main properties of Collection, such as inverse, cascade, focusing on sample code.

One To One Mapping

When there is one-to-one mapping between tables, each entity class is defined and the relationship between classes processed as one-to-one mapping.

Sample Source

```
@Entity
public class Employee {
    @One-to-one
    private TravelProfile profile;
}

@Entity
public class TravelProfile {
    @One-to-one
    private Employee employee;
}
```

Above example shows that Employee and TravelProfile are mapped indicating the Annotation of One-to-one, respectively.

One To Many Mapping

Where there is one-to-many mapping, each entity class is defined and indicate its relationship by writing one-to-many mapping and many-to-one mapping annotations.

Sample Source

```
@Entity
public class Department{
    @One-to-many(targetEntity=User.class)
    private Set<User> users = new HashSet(0);
}

@Entity
public class User{
    @Many-to-one
    private Department department;
}
```

Above example shows that it has Department:User = 1:N relationship indicates One-to-many in Department class for that relation, and many-to-one in User class.

Collection Type

Collection can use the ListMap other than the Set used in above examples. Following is the method.

Set

It is defined using <set> as the java.util.Set type. The order of saving the object cannot be known and the duplicate saving of same object is not allowed. (use HashSet) Following is the example of source code defining the collection object using the set tag.

Sample Source

```
@Entity
public class Department{
    @One-to-many(targetEntity=User.class)
    private Set<User> users = new HashSet(0);
}
```

List

It is defined using <list> as the java.util.Set type. The List type shows the order of stored objects, to store the order of storage on the table, it requires to define index column separately (use ArrayList). Following is the example of source code defining the collection object using the list tag.

Sample Source

```
@Entity
public class Department{
    @One-to-many(targetEntity=User.class )
    private List<User> users = new ArrayList(0);
}
```

Map

It is defined using <map> as the java.util.Set type using (key and value). (use HashMap) Following is the example of source code defining collection object using map tag. Annotation of MapKey should be additionally defined for key setting.

Sample Source

```
@Entity
public class Department{
    @One-to-many(targetEntity=User.class)
    @MapKey(name="userId")
    private Map<String,User> users ;
}
```

Conjunctions and Disjunctions

In deciding one-to-many (parent and child) relationship, it is defined as disjunction relationship if there is a reference from the child to parent, and conjunction where there is no reference.

Conjunctions

It is defined without reference of parent entity on the child entity.

Sample Source

```
@Entity
public class Department{
    @One-to-many(targetEntity=User.class)
    private Set<User> users = new HashSet(0);
}
```

```
@Entity
public class User{
    @Column(name="DEPT_ID")
    private String deptId;
}
```

Above example shows that reference for department class was not designated in User class, but depId was designated simply in mapping with column DEPT_ID of table.

Disjunctions

It is defined by designating the reference of parent entity on the child entity.

Sample Source

```
@Entity
public class Department{
    @One-to-many(targetEntity=User.class)
    private Set<User> users = new HashSet(0);
}

@Entity
public class User{
    @Many-to-one
    private Department department;
}
```

Above example shows that mapping relation was designated through Many-to-one Annotation for the Department class in User class.

mappedBy,Cascade Property

mappedBy and cascade are one of properties that have important meaning at the time of Collection definition and have the following meaning.

- mappedBy: properties for defining option on which entity will perform relation connection between objects.
- cascade : properties for defining the option on whether to transfer CUD for parent object to the child object too.

mappedBy: ○, cascade: ○

If using by defining both mappedBy and cascade, perform relation connection in child Entity and automatically perform child entity at the time of CUD processing in parent Entity.

Define Source

```
@Entity
public class Department{
    @One-to-many(targetEntity=User.class,mappedBy="deptId",cascade={CascadeType.PERSIST,
CascadeType.MERGE})
    private Set<User> users = new HashSet(0);
}

@Entity
public class User{
    @Many-to-one(targetEntity=Department.class)
    private Department department;
}
```

Sample Source

```
// Relation setting through setDepartment method of User(child) Entity
user.setDepartment(department);

// Simultaneous processing up to child with saving of parent entity
em.persist(department);
```

mappedBy: ○, cascade: X

If using by defining mappedBy only, perform relation connection at child Entity and perform CUD at child and parent.

Define Source

```
@Entity
public class Department{
    @One-to-many(targetEntity=User.class,mappedBy="deptId")
    private Set<User> users = new HashSet(0);
}

@Entity
public class User{
    @Many-to-one(targetEntity=Department.class)
    private Department department;
}
```

Sample Source

```
// Set relation through setDepartment method of User(child) Entity
user.setDepartment(department);

// respectively processing parent/child entity
em.persist(department);
em.persist(user);
```

mappedBy: X, cascade: ○

If using by defining cascade only, perform relation connection at parent Entity and automatically process child Entity at parent Entity at the time of CUD processing.

Define Source

```
@Entity
public class Department{
    @One-to-many(targetEntity=User.class,cascade={CascadeType.PERSIST, CascadeType.MERGE})
    private Set<User> users = new HashSet(0);
}

@Entity
public class User{
    @Many-to-one(targetEntity=Department.class)
    private Department department;
}
```

Sample Source

```
// Set relation through getUsers().add method of Department(parent) Entity
department.getUsers().add(user);

// Simultaneous processing up to child with saving of parent entity
em.persist(department);
```

mappedBy: X, cascade: X

If not defining all, perform relation connection at parent Entity and perform CUD at parent and child respectively.

Define Source

```
@Entity
public class Department{
    @One-to-many(targetEntity=User.class)
    private Set<User> users = new HashSet(0);
}

@Entity
public class User{
    @Many-to-one(targetEntity=Department.class)
    private Department department;
}
```

Sample Source

```
// Set relation through getUsers().add method of Department(parent) Entity
department.getUsers().add(user);

// respectively processing parent/child entity
em.persist(department);
em.persist(user);
```

Many To Many Mapping

If there is M:N mapping between table, define respective Entity class and indicate Annotation of ManyToMany on both sides to show relation.

Sample Source

```
@Entity
public class Role{
    @ManyToMany(targetEntity=User.class)
    private Set<User> users = new HashSet(0);
}

@Entity
public class User{
    @ManyToMany
    @JoinTable(name="AUTHORITY",
                joinColumns=@JoinColumn(name="USER_ID"),
                inverseJoinColumns=@JoinColumn(name="ROLE_ID"))
    private Set<Role> roles = new HashSet(0);
}
```

Above example shows relation of Role:User = M:N and indicates such relations as ManyToMany in Role class. Indicate as ManyToMany in User class to show relation and define separate table for relation in User class. In this case, it is known that AUTHORITY is used as a relation table connecting ROLE and USER.