Creating and Using String-Based Criteria Queries

This chapter describes how to create weakly-typed string-based Criteria API queries.

Overview of String-Based Criteria API Queries

String-based Criteria API queries ("string-based queries") are Java programming language queries that use strings rather than strongly-typed metamodel objects to specify entity attributes when traversing a data hierarchy.

String-based queries are constructed similarly to metamodel queries, can be static or dynamic, and can express the same kind of queries and operations as strongly-typed metamodel queries.

Strongly-typed metamodel queries are the preferred method of constructing Criteria API queries.

The main advantage of string-based queries over metamodel queries is the ability to construct Criteria queries at development time without the need to generate static metamodel classes or otherwise access dynamically generated metamodel classes.

The main disadvantage to string-based queries is their lack of type safety, which may lead to runtime errors due to type mismatches that would be caught at development time when using strongly-typed metamodel queries.

For information on constructing criteria queries, see Chapter 35, Using the Criteria API to Create Queries

Creating String-Based Queries

To create a string-based query, specify the attribute names of entity classes directly as strings, rather than the attributes of the metamodel class.

For example, this query finds all `Pet` entities where the value of the `name` attribute is `Fido`:

```
CriteriaQuery<Pet> cq =
cb.createQuery(Pet.class);
Root<Pet> pet = cq.from(Pet.class);
cq.where
(cb.equal(pet.get("name"),"Fido"));
...
```

The name of the attribute is specified as a string.

This query is the equivalent of the following metamodel query:

```
CriteriaQuery<Pet> cq =
cb.createQuery(Pet.class);
Metamodel m = em.getMetamodel();
EntityType<Pet> Pet_ =
m.entity(Pet.class);
Root<Pet> pet = cq.from(Pet.class);
cq.where
(cb.equal(pet.get(Pet_.name),"Fido"));
```

Note - Type mismatch errors in string-based queries won't appear until the code is executed at runtime, unlike the above metamodel query, where type mismatches will be caught at compile time.

Joins are specified in the same way:

```
CriteriaQuery<Pet> cq =
cb.createQuery(Pet.class);
```

```
Root<Pet> pet = cq.from(Pet.class);
Join<Owner, Address> address =
pet.join
("owners").join("addresses");
...
```

All the conditional expressions, method expressions, path navigation methods, and result restriction methods used in metamodel queries can be used in string-based queries.

In each case, the attributes are specified using strings.

For example, here is a string-based query that uses the `in` expression:

```
CriteriaQuery<Pet> cq =
cb.createQuery(Pet.class);
Root<Pet> pet = cq.from(Pet.class);
cq.where
(pet.get("color").in("brown", "black"));
```

Here is a string-based query that orders the results in descending order by date:

```
CriteriaQuery<Pet> cq =
cb.createQuery(Pet.class);
Root<Pet> pet = cq.from(Pet.class);
cq.select(pet);
cq.orderBy
(cb.desc(pet.get("birthday")));
```

Executing String-Based Queries

String-based queries are executed similarly to strongly-typed Criteria queries.

First create a `javax.persistence.TypedQuery` object by passing the criteria query object to the `EntityManager.createQuery` method and then call either `getSingleResult` or `getResultList` on the query object to execute the query.

```
CriteriaQuery<Pet> cq =
cb.createQuery(Pet.class);
Root<Pet> pet = cq.from(Pet.class);
cq.where
(cb.equal(pet.get("name"),"Fido"));
TypedQuery<Pet> q =
em.createQuery(cq);
List<Pet> results =
q.getResultList();
```