Cougaar

Cougaar Agent Architecture Open-Source site

Query Annotations

New in Cougaar 12.6 are Query Annotations.

Query annotations are a convenient way to do a filtered query of the blackboard for a collection of objects.

The idea of Query Annotations is that a plugin developer writes a method that does some arbitrary operation on a specified kind of object, optionally matching a given predicate. Invoking the `runQuery` method then has the effect of invoking this method on each instance returned by the query that uses the predicate, returning the Collection of objects that matched the query.

The plugin class must extend `org.cougaar.core.plugin.AnnotatedSubscriptionsPlugin`.

The definition of the annotation is:

```
public @interface Query {
    String name() default "";
    String where() default NO_VALUE;
}
```

In using the annotation, if a `name` argument is not provided, or is provided as `""`, the name will be taken from the name of the method being annotated. The query will always filter by the type of the first argument of the annotated method. For further filtering you can provide the name of boolean-valued method with one argument as the `where` argument. The type of that argument must match the type of first argument of the annotated query method.

Attach the `@Cougaar.Query()` annotation to a public method with at least one argument to run the body of the method on each instance matching a blackboard query that uses the given predicate method. Invoking such a query would generally be done with the `runQuery` method in `AnnotatedSubscriptionsPlugin` passing the name as the argument. Contextual arguments can be included as varargs.

Any positive number of arguments can be provided to the method that has the `@Cougaar.Query()` annotation. The first argument is used to filter the type match in the query. The remaining arguments can be anything you want but must match the `runQuery` call (see below).

You can have as many Query annotations as you want, regardless of argument type, as long as the query names are unique within the component.

Example annotated method:

```
@Cougaar.Query(name="queryMoreThanThree", where="isMoreThanThree")
public void helloQuery(HelloObject match, QueryStats context) {
    ++context.count;
}
```

Example predicate method (note how the name of this method is the value of `where` argument above, and the data type of the first argument of both functions are the same):

```
public boolean isMoreThanThree(HelloObject hello) {
    return hello.getValue() > 3;
}
```

Example more complex annotated method and its predicate method:

```
@Cougaar.Query(name="complexQueryMoreThanThree",
                    where="complexIsMoreThanThree")
public void complexHelloQuery(HelloObject match, QueryStats context,
                              Integer moreContext) {
    ++context.count;
}

public boolean complexIsMoreThanThree(HelloObject hello,
                                      Integer moreContext) {
    return hello.getValue() > 3 && moreContext == 0;
}
```

The `runQuery` method invokes the query method on each instance returned by the query that uses the predicate. It returns the same raw Collection returned by the blackboard query itself, which is the Collection of objects that matched the query. *Note* that arguments to `runQuery` are not checked until runtime, which is an inherent restrictions of Java annotations.

Definition of the `runQuery` method:

```
/**
  * Run the query with the given id and context
  *
  * @param id
  *           Name of the query.
  * @param type
  *           The type of the first argument of the annotated
  *           method, which is also the element type of the
  *           returned Collection of matches.
  * @param queryContext
  *           Optional data that preserves context
  *           between iterations.
  * @return   The blackboard query result, or an empty list if
  *           the id doesn't match any known query.
  */
public Collection runQuery(String id, Class type,
                           Object... queryContext)
```

A complete plugin example:

```
package org.cougaar.demo.hello;

import java.util.Collection;
import org.cougaar.core.plugin.AnnotatedSubscriptionsPlugin;
import org.cougaar.util.annotations.Cougaar;

/**
 * Illustrate query annotation
 */
public class HelloQueryPlugin extends AnnotatedSubscriptionsPlugin {
   /**
    * Named blackboard query.
    * When the query is invoked via {@link #runQuery},
    * this method will be called once per match.
    * The where clause defines the match predicate.
    * Any number of context variables are passed to
    * the processing method. The same context is passed
    * in each iteration.
    *
    * @param match The next matching blackboard object.
    *
    * @param context for the query.
    */
   @Cougaar.Query(name="queryMoreThanThree", where="isMoreThanThree")
   public void helloQuery(HelloObject match, QueryStats context) {
      ++context.count;
   }
```

```
    }

    /**
     * Predicate to match against all HelloObject on the blackboard
     * @param hello
     * @return
    */
    public boolean isMoreThanThree(HelloObject hello) {
       return  hello.getValue() > 3;
    }

    @Override
    protected void execute() {
       super.execute();
       QueryStats stats = new QueryStats();
       Collection result = runQuery("queryMoreThanThree",
                                    HelloObject.class, stats);
       log.shout(stats.count
                 + " Hello messages processed and "
                 + result.size() + " results returned");
    }

    /**
     * A context state that can change by each match invocation.
     */
    private static final class QueryStats {
       private int count;
    }
}
```