Queryll:Java database Queries Through Bytecode Rewriting

Wu Cong
2010.02.09

Introduction-

- Middleware system
- Interfacing java with SQL databases by providing database query facilities to java
- Standard java syntax; no special compiler IDE
- Queryll bytecode rewriter replaces the code with equivalent SQL queries

Related Work

- Other middleware systems use special programming languages to interface java to other programming languages . All require the use custom languages to access their features
- Weak points: have to learn new language; have to write code between models; have to embed queries in strings that are not error-checked; have to marshal parameters into special data structures

Related Work-

- Hybrid programming language, SQLJ
- Strong points: error checking, automatic data marshaling
- Weak points: new compiler, new IDE; not applicable with multiple interface languages; tightly bound to SQL table-oriented view of data

Related Work

- Standard database middleware layer JDBC
- Queries in strings passed through API
- Weak points: must manually pack parameters into queries and manually read out and interpret individual fields from results; SQL table-oriented view of data

Related Work-

- ORM Tools (Hibernate, EJB)
- Strong points: specify a mapping from SQL tables to an object representation; no data marshaling issue
- Weak points: can not be used for complex queries

Related Work

- LINQ
- Support lambda expressions

Goal

- No new language, just API
- No new compilers ,no data marshaling, no embedding code inside strings
- Use bytecode rewriting

Bytecode rewriting and decompiling

- Compilers compile java into bytecode, stored in classfiles, which can be executed using a java VM
- Other bytecode rewriting techniques: J-Orchestra, aspect-oriented programming tools, some ORM tools
  - Weak point: only modifying surface features of the code
- QueryII borrows techniques from classfile decompilation

QueryII with QueryII

- Designed to conform with standard Java patterns for working collections
  - Cannot convert arbitrary Java to SQL
- Use ORM to allow database entities to be represented and manipulated as objects in JAVA
- Support selections, projections, joins
- No support for aggregation or nested queries; no support for SQL ordering and limit operation

QueryII ORM

- Use a custom light-weight ORM tool to map tables to classes
- Generate the classes for each entity with accessor methods for fields and special methods for traversing relationships between objects
- Create a Entity manager responsible for ensuring the database data and in-memory object representations remain consistant.

Simple Queries and Selection

- For-each loop over collections called QuerySet
- Loop = query; execution fills an output QuerySet with the results
- Loop must iterate over all original QuerySet elements, and only add elements to the new QuerySet

Example

- QuerySet<String> canadian=new QuerySet<String>;
- String country= "Canada";
- For (Client c:em.allClient())
  - If (c.getCountry().equals(country))
    - Canadian.add(c.getname());

Projection-

- Pair object

```
QuerySet<Pair<Account, Double>> overdrawn
      = new QuerySet<Pair<Account, Double>>();
for (Account a: em.allAccount()) {
   if (a.getBalance() < a.getMinBalance()) {
      double penalty = (a.getMinBalance() - a.getBalance()) * 0.001;
      overdrawn.add(new Pair<Account, Double>(a, penalty);
   }
}
```

Join

```
QuerySet<Pair<Client, Account>>
   swiss1 = new QuerySet<Pair<Client, Account>>(),
   swiss2 = new QuerySet<Pair<Client, Account>>();

for (Account a: em.allAccount())
   if (a.getHolder().getCountry().equals("Switzerland"))
      swiss1.add(new Pair<Client, Account>(a.getHolder(), a));

for (Client c: em.allClient())
   if (c.getCountry().equals("Switzerland"))
      swiss2.addAll(Pair.PairCollection(c, c.getAccounts());
```

Implementation-

- Labelled @Query annotation
- use Sable's Soot framework Jimple code (execution stack)

```
[ for (Office of: em.allOffice()) {
     if (of.getName().equals("Seattle"))
        westcoast.add(of);
     else if (of.getName().equals("LA"))
        westcoast.add(of);
  }
```

Example

```
          1:  $r12 = r1.<EntityManager: Set allOffice()>();
          2:  r8 = $r12.<Set: Iterator iterator()>();
          3:  goto label3;

label4:   4:  $r13 = r8.<Iterator: Object next()>();
          5:  r14 = (Office) $r13;
          6:  $r15 = r14.<Office: String getName()>();
          7:  $z0 = $r15.<String: boolean equals(Object)>("Seattle");
          8:  if $z0 == 0 goto label2;

          9:  r11.<Set: boolean add(Object)>(r14);
         10:  goto label3;

label2:  11:  $r16 = r14.<Office: String getName()>();
         12:  $z1 = $r16.<String: boolean equals(Object)>("LA");
         13:  if $z0 == 0 goto label3;

         14:  r11.<Set: boolean add(Object)>(r14);

label3:  15:  $z2 = r8.<Iterator: boolean hasNext()>();
         16:  if $z2 != 0 goto label1;
```

Second step-

- Identify loops within the code
- Only goto statements to describe control flow. Two ways
- Analyze as a whole; restructure back to loops
  - Use standard graph algorithms to identify loops