Wprowadzenie

db4o

Norbert Potocki

1 czerwca 2009

Wprowadzenie

- db4o = db4objects = database for objects
- w pełni obiektowa baza danych
- podwójna licencja - GPL oraz komercyjna
- dostępna dla Java oraz .NET
- wspierana przez korporację Versant - szybki rozwój
- olbrzymie środowisko użytkowników (ponad 60 000) aktywnie uczestniczących w rozwoju bazy i jej testowaniu
- wersja stabilna - 7.4
- wersja rozwojowa - 7.10

Zalety ODBMS w stosunku do RDBMS

- prostota użycia (zapytania piszemy w języku natywnym dla przechowywanych danych)
- bezpośrednie odwzorowanie przechowywanych danych (obiektów) w bazie
- brak konieczności wprowadzania dodatkowej warstwy ORM (np. Hibernate)
- łatwe przeglądanie zawartości bazy
- lepsza wydajność w “czysto obiektowych” zapytaniach

Wady ODBMS w stosunku do RDBMS

- na chwilę obecną gorsze wsparcie dla tej technologii
- gorsza wydajność w niektórych zastosowaniach

Model danych

```
public class Pilot {
    private String name;
    private int points;

    public Pilot(String name,int points) {
        this.name=name;
        this.points=points;
    }

    public int getPoints() {
        return points;
    }

    public void addPoints(int points) {
        this.points+=points;
    }

    public String getName() {
        return name;
    }

    public String toString() {
        return name+"/"+points;
    }
}
```

Używnie bazy - tryb wbudowany

```
ObjectContainer db = Db4oEmbedded.openFile(Db4oEmbedded
        .newConfiguration(), DB4OFILENAME);
try {

    Pilot pilot1 = new Pilot("Michael Schumacher", 100);
    db.store(pilot1);

    Pilot pilot2 = new Pilot("Rubens Barrichello", 99);
    db.store(pilot2);

} finally {
    db.close();
}
```

Używnie bazy - tryb rozproszony

- musimy implementować własny serwer..
- .. ale jest to proste

```
ObjectServer server = Db4oClientServer.openServer(Db4oClientServer
        .newServerConfiguration(), DB4OFILENAME, 0);

try {
    ObjectContainer client1 = server.openClient();
    ObjectContainer client2 = server.openClient();
    // ...
    client1.close();
    client2.close();
} finally {
    server.close();
}
```

```
    objectContainer =
      Db4oClientServer.openClient(Db4oClientServer
      .newClientConfiguration(), HOST, PORT, USER, PASS);
```

Rodzaje zapytań - Query By Example (QBE)

- przydatne tylko w bardzo prostych aplikacjach
- wszystkie obiekty muszą zostać wczytane do pamięci i przejrzane
- problemy z wyszukaniem obiektów o wartościach pól odpowiadających wartościom domyślnym

```
Pilot proto = new Pilot(null, 0);
ObjectSet result = db.queryByExample(proto);

ObjectSet result = db.queryByExample(Pilot.class);
List <Pilot> pilots = db.query(Pilot.class);

Pilot proto = new Pilot("Michael Schumacher", 0);
ObjectSet result = db.queryByExample(proto);

Pilot proto = new Pilot(null, 100);
ObjectSet result = db.queryByExample(proto);
```

Rodzaje zapytań - Native Queries (NQ)

- działają szybko
- bardzo elastyczne
- optymalizowane przed wykonaniem
- weryfikacja na poziomie kompilacji

```
List <Pilot> result = db.query(new Predicate<Pilot>() {
      public boolean match(Pilot pilot) {
          return pilot.getPoints() > 99
              && pilot.getPoints() < 199
              || pilot.getName().equals("Rubens Barrichello");
    }
});
```

Rodzaje zapytań - SODA Query API

- bezpośredni dostęp do wierzchołków grafu zapytania
- brak weryfikacji typów - odwołania do pól poprzez ich nazwy w postaci napisów

```
Query query=db.query();
query.constrain(Pilot.class);
ObjectSet result=query.execute();

Query query=db.query();
query.constrain(Pilot.class);
query.descend("name").constrain("Michael Schumacher");
ObjectSet result=query.execute();

Query query=db.query();
query.constrain(Pilot.class);
Constraint constr=query.descend("name")
        .constrain("Michael Schumacher");
query.descend("points")
        .constrain(new Integer(99)).and(constr);
ObjectSet result=query.execute();
```

Aktualizacja i usuwanie obiektów

- musimy wywołać wyszukanie obiektu, bo inaczej db4o przyjmie, że dodajemy nowy obiekt o identycznych wartościach pól jak obiekt już istniejący w bazie

```
ObjectSet result = db
        .queryByExample(new Pilot("Michael Schumacher", 0));
Pilot found = (Pilot) result.next();
found.addPoints(11);
db.store(found);

ObjectSet result = db
        .queryByExample(new Pilot("Michael Schumacher", 0));
Pilot found = (Pilot) result.next();
db.delete(found);
```

Zagnieżdżone obiekty

- pobieranie zagnieżdżonych obiektów następuje tylko do pewnego poziomu
- domyślnie zagnieżdżone obiekty nie są aktualizowane w bazie
- usuwanie obiektu nie powoduje skasowania jego dzieci
- podczas usuwania obiektu nie następuje sprawdzenie, czy istnieją do niego jeszcze jakieś referencje

```
EmbeddedConfiguration config = Db4oEmbedded.newConfiguration();
config.common().objectClass("com.db4o.f1.chapter2.Car").cascadeOnUpda
te(true);
ObjectContainer db = Db4oEmbedded.openFile(config, DB4OFILENAME);

EmbeddedConfiguration config = Db4oEmbedded.newConfiguration();
config.common().objectClass("com.db4o.f1.chapter2.Car").cascadeOnDele
te(true);
ObjectContainer db = Db4oEmbedded.openFile(config, DB4OFILENAME);
```

Kolekcje, tablice, interfejsy i dziedziczenie

- obsługa kolekcji, tablic, interfejsów i dziedziczenia jest zrobiona w sposób "naturalny"

```
public class SensorReadout {
  ...
}

public class TemperatureSensorReadout extends SensorReadout {
  ...
}

public class PressureSensorReadout extends SensorReadout {
  ...
}

SensorReadout proto=
    new TemperatureSensorReadout(null,null,null,0.0);
ObjectSet result=db.queryByExample(proto);

SensorReadout proto=new SensorReadout(null,null,null);
ObjectSet result=db.queryByExample(proto);

ObjectSet result=db.queryByExample(SensorReadout.class);
```

Aktywacje

- pobieranie zagnieżdżonych obiektów następuje tylko do pewnego poziomu (domyślnie 5)
- konfigurowalne na różne sposoby

```
    public static void
retrieveSnapshotsSequentiallyImproved(ObjectContainer db) {
        ObjectSet result = db.queryByExample(Car.class);
        Car car = (Car) result.next();
        SensorReadout readout = car.getHistory();
        while (readout != null) {
            db.activate(readout, 1);
            System.out.println(readout);
            readout = readout.getNext();
        }
    }
```

Transakcyjność

- każde połączenie jest oddzielną transakcją
- tryb – “read commited”
- wykonanie rollback nie przywraca poprzedniego stanu obiektów w aplikacji – jeśli chcemy tego dokonać, to musimy pobrać je ponownie z bazy

```
Pilot pilot=new Pilot("Rubens Barrichello",99);
Car car=new Car("BMW");
car.setPilot(pilot);
db.store(car);
db.commit();

Pilot pilot=new Pilot("Michael Schumacher",100);
Car car=new Car("Ferrari");
car.setPilot(pilot);
db.store(car);
db.rollback();
```

Inne mechanizmy

- automatyczne aktywacje
  - pobiera całe drzewo obiektów
  - mało wydajne
- implementacja interfejsu Activatable
  - atrybuty obiektu są automatycznie aktywowane w momencie ich pierwszego użycia
- przezroczyste utrwalanie
  - obiekty muszą implementować interfejs Activatable
  - lista zmienionych obiektów jest zapamiętywana
  - przy operacji commit obiekty z listy są zapisywane w bazie

```
Pilot pilot=new Pilot("Rubens Barrichello",99);
Car car=new Car("BMW");
car.setPilot(pilot);
db.store(car);
db.commit();

Car car=new Car("Ferrari");
car.setPilot(pilot);
db.store(car);
db.rollback();
```

Przydatne narzędzia

- wtyczki do NetBeans oraz Eclipse do przeglądania bazy
- OME (Object Manager Enterprise) — dostępny od wersji 7.8
- wtyczki do łączenia z innymi językami i technologiami:
  - Spring
  - Ruby
  - JRuby
  - Silverlight

Przydatne materiały

- The Definitive Guide to db4o — Stefan Edlich, Henrik Hörning, Reidar Hörning, Jim Paterson, Apress 2006
- db4o Tutorial — na stronie projektu
- http://db4o.com/ — główna strona projektu
- http://developer.db4o.com/ — strona dla deweloperów