Objects, Relations, and everything

Jonathan Edwards
9/17/2012
Design problem: connect a program to a relational database
Query languages were designed as a user interface

```
SELECT ID, COST
FROM WIDGETS WHERE
COST < 100
```

<table>
<thead>
<tr>
<th>ID</th>
<th>COST</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>99</td>
</tr>
<tr>
<td>4</td>
<td>29</td>
</tr>
</tbody>
</table>
Solution:
program emulates a user

printf(
"SELECT ID, COST FROM
WIDGETS WHERE COST < 100"
)

parse("1 99\n4 29")
Connection con = ...;
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery("SELECT ID, COUNT FROM SITES");
while (rs.next()) {
    String id = rs.getString("ID");
    int count = rs.getInt("COUNT");
    System.out.println(id + ": " + count);
}
Solution: ORM (Object Relational Mapping)

- Programmer thinks in terms of objects; doesn't need to know SQL or relational database theory
- Object heap caches state of DB

<table>
<thead>
<tr>
<th>sites</th>
<th></th>
<th>sites</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>id</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;somesite&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;anothersite&quot;</td>
</tr>
</tbody>
</table>

 ORM
Java Hibernate

SQL

CREATE TABLE sites(
    id VARCHAR,
    count INT
)

public class Site {
    private String id;
    private int count;

    public Event() {} 

    public String getId() {
        return id;
    }
    private void setId(String id) {
        this.id = id;
    }

    public int getCount() {
        return count;
    }
    public void setCount(int count) {
        this.count = count;
    }
}
Site.hbm.xml

```xml
<hibernate-mapping>
  <class name="Site" table="sites">
    <id name="id" type="string"/>
    <property name="count" type="int"/>
  </class>
</hibernate-mapping>
```
@Entity
@Table
public class Site {
    private String id;
    private int count;

    public Event() {}

    @Id
    public String getId() {
        return id;
    }
    private void setId(String id) {
        this.id = id;
    }

    @Column
    public int getCount() {
        return count;
    }
    public void setCount(int count) {
        this.count = count;
    }
}
Session session = sessionFactory.openSession();
session.beginTransaction();
s = new Site();
s.setId("mysite");
s.setCount(1);
session.save(s);
session.getTransaction().commit();
session.close();
@Entity
@Table
public class Site {
    private String id;
    private int count;

    public Event() {}  

    @Id
    public String getId() {
        return id;
    }
    private void setId(String id) {
        this.id = id;
    }

    @Column
    public int getCount() {
        return count;
    }
    public void setCount(int count) {
        this.count = count;
    }
}
Rails Active Record

rails generate model Site id:string count:integer
rake db:migrate

db/schema.rb

ActiveRecord::Schema.define(:version => 0172…) do
  create_table "sites", :force => true do |t|
    t.integer "count"
    t.datetime "created_at", :null => false
    t.datetime "updated_at", :null => false
  end
end
class Site < ActiveRecord::Base
end

@Entity
@Table
public class Site {
    private String id;
    private int count;

    public Event() {}

    @Id
    public String getId() {
        return id;
    }
    private void setId(String id) {
        this.id = id;
    }

    @Column
    public int getCount() {
        return count;
    }
    public void setCount(int count) {
        this.count = count;
    }
}
Site.instance_methods  # => 309 methods!
s = Site.new
s.count = 1

s.count=(1)

s.method_missing('count=', [1])

s.class_eval(
  "def count=(v);
    write_attribute('count', v);
  end"
)

s.send('count=', [1])
More features

- Associations
- Queries using normal Ruby syntax
- Declarative validations
- Validation callbacks
- Migrations (key innovation)
Ruby is for meta

• Add methods on the fly to objects and classes
  – Monkeypatch built-ins like String and Kernel
  – Intercept methods and wrap with other behavior

• Class definitions executed at load, so class methods can act like macros
  – E.g. attr_accessor

• No fixed definitions
  – Change anything anyway anytime
Metaprogramming Ruby

Program Like the Ruby Pros

Paolo Perrotta

Edited by Jill Steinberg

The Facets of Ruby Series
Philosophy of Active Record

• a coherent wrapper as a solution for the inconvenience that is object-relational mapping
• infer complex relations and structures from minimal explicit direction
• Convention over Configuration:
  – No XML-files!
  – Lots of reflection and run-time extension
  – Magic is not inherently a bad word
• Admit the Database:
  – Lets you drop down to SQL for odd cases and performance
  – Doesn’t attempt to duplicate or replace data definitions

http://api.rubyonrails.org/files/activerecord/README_rdoc.html
ORM disadvantages

• Database differences not entirely masked
• Subtle concurrency bugs
  – E.g. anachronistic objects
• Still need to know SQL for performance tuning and special cases
• Leaky abstraction
• Big pile of code & docs
Beyond ORM

- NoSQL: API-centric DB's
- Back to SQL: anorm
- Open research problem: unify DB & PL
  - impedance mismatch problem
- No matter what, RDBs will remain
  - Software doesn't die!
  - It just gets hidden beneath another layer