Acitve Record

PoEAA - 10. Data Source Architectual Patterns

@vividmuimui

2017/08/24 PoEAA読書会資料

Active Recordといえば

RailsののActiveRecord。これも、このActiveRecordパターンを適用したもの。

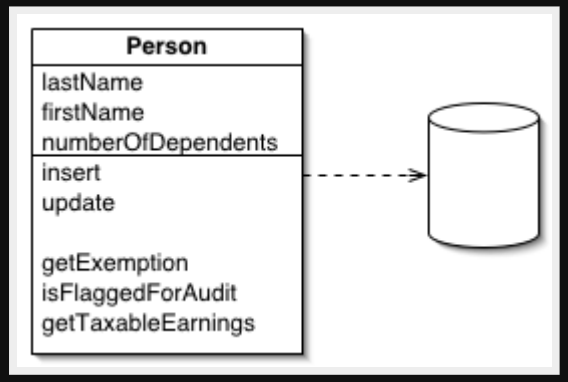
「10. Data Source Architectual Patterns」には以下の4種類があるが、その中で一番シンプルなパターン。

- Table Data Gateway
- Row Data Gateway
- Active Record
- Data Mapper

すごく雑に言うと

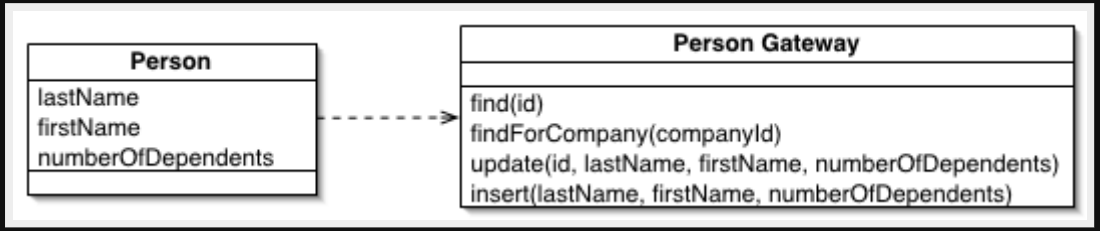
- DBのテーブル/ビューの1つのレコードを表現し、DBへのアクセスするロジック
- ドメインロジック

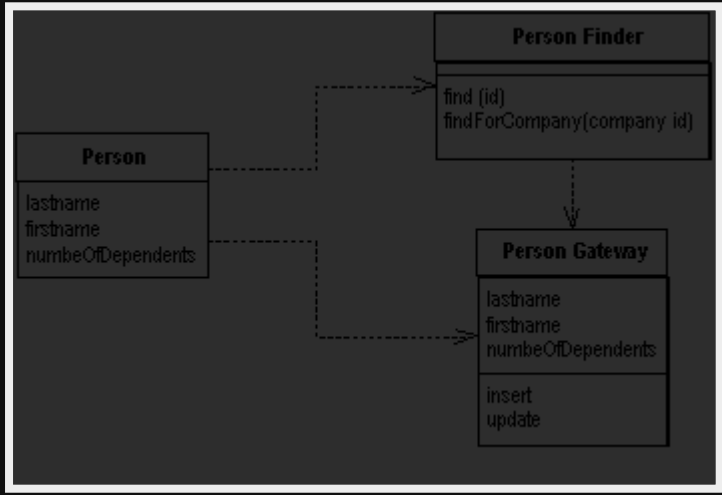
この両方を１つのオブジェクトに詰め込んだパターンで、4つのパターンのうちとてもシンプルなパターン。

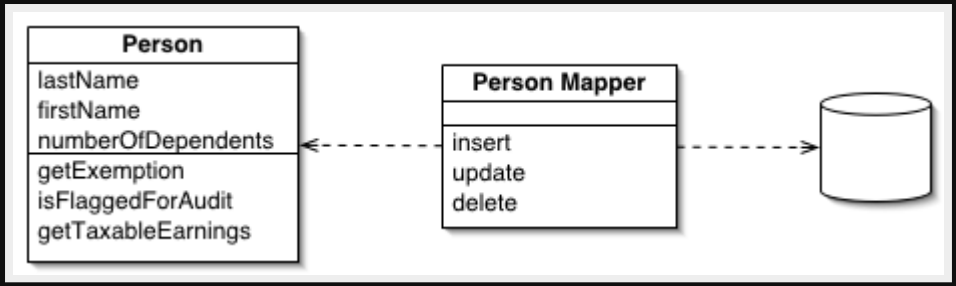


(https://www.martinfowler.com/ より)

ほかのパータンはちょっと複雑







(https://www.martinfowler.com/ より)

どのように動くか(How It Works)

1️⃣

- ActiveRecordは、DBのレコードに密接している
- ActiveRecordは、DBへの読み書きをする責任をもっていて、ドメインロジックも含んでいる
- ActiveRecordのデータ構造(各フィールド)は、DBの1つのレコードのカラムと一致する
  - nameカラムを持っていれば、ActiveRecordはnameフィールドを持つ
- Foreign keyもそのまま使える
- ActiveRecordは、DBのテーブルだけでなくViewにも適用できるが、updateは困難なのでreadのみに使うのがよい

どのように動くか(How It Works)2️⃣

基本的には次のようなメソッドを持つ

- selectのSQLの結果を用いてActiveRecordインスタンスを生成できる
- 新しいActiveRecordインスタンスを生成してinsertが出来る
- 検索のためのクラスメソッドがある
  - ActiveRecordのインスタンス(s)を返す
- ActiveRecordのインスタンスのデータで、update, insertが出来る
- ActiveRecordの各フィールドのget, setが出来る
- ドメインロジック

どのように動くか(How It Works)

3️⃣

このパターンは便利だが、DBが存在することを隠蔽できていない
As a result you usually see fewer of the other object -relational mapping patterns present when you’re using ActiveRecord.

ActiveRecordは、RowDataGatewayパターンとよく似ている
RowDataGatewayはデータベースアクセスしか持たないが、ActiveRecordはデータベースアクセスとドメインロジックを持っている

このパターンでは、find系のクラスメソッドが多くなる傾向にあるが、ほかクラスに分けてはいけないということはない
ほかのパターンと併用しつつ、ActiveRecordはViewやQuery用に使うのも良い

いつ使うのか(When to Use It) 1️⃣

Active Recordは単なるCRUDするだけのようなドメインロジックが複雑すぎないケースではいい選択肢になる
１つのレコードに対する、Derivations and validations(データの取得と検証?)がうまく出来る

いつ使うのか(When to Use It) 2️⃣

Domain Modelの設計時の主な選択肢として、
ActiveRecordとDataMapperがあがるが、ActiveRecordはシンプルで理解しやすく作るのが簡単なのがメリット

ドメインロジックが複雑になってくると、relationships, collections, inheritanceなどがすぐに欲しくなるだろう
それらをActiveRecordに適用するのは難易度が高いので、DataMapperを使ったほうがよい

いつ使うのか(When to Use It) 3️⃣

ActiveRecordの短所の１つに、オブジェクトデザインとDBデザインが密結合しているため、プロジェクトの成長に合わせてリファクタしてくのが難しいという点がある

サンプル通りに実装してみた

javaのサンプルが本に載っているので、rubyで実装してみた
(DBのコネクション部分はrailsのActiveRecordに頼っちゃいました 😝)

まずは準備 1️⃣

```
# Gemfile
source 'https://rubygems.org'

gem "activerecord"
gem "sqlite3"
```

準備 2️⃣

```ruby
require "active_record"

class CreatPeoples < ActiveRecord::Migration[5.1]
  def change
    create_table :people do |t|
      t.string :lastname
      t.string :firstname
      t.integer :number_of_dependents
    end
  end
end

CreatPeoples.migrate(:up)
```

peopleテーブルは、
id, lastname, firstname, number_of_dependents
の4つのカラムを持っている

準備 3️⃣

```
ActiveRecord::Base.logger = Logger.new(STDOUT)
ActiveRecord::Base.establish_connection(
  adapter: "sqlite3",
  database: "demo.sqlite3"
)
```

sqliteにつなげるように設定する

これで準備は完了

まずは .find の実装

```
class People
  FIND_SQL = "SELECT id, lastname, firstname, number_of_dependents FROM people WHERE id = ?;"

  class << self
    # キャッシュ周りのロジックは省略しました
    def find(id)
      # SQLをいい感じに組み立てて実行
      row_hash = con.select_one(ActiveRecord::Base.send(:sanitize_sql_array, [FIND_SQL, id]))
      new(row_hash.symbolize_keys) if row_hash # レコードが見つかったら自身のインスタンスを作成する
    end

    def con
      @con ||= ActiveRecord::Base.connection
    end
  end

  attr_accessor :id, :lastname, :firstname, :number_of_dependents

  def initialize(id: nil, lastname: nil, firstname: nil, number_of_dependents: nil)
    @id, @lastname, @firstname, @number_of_dependents = id, lastname, firstname, number_of_de
  end
end
```

実行してみると

```
(main)> People.find(1)
(0.2ms)  SELECT id, lastname, firstname, number_of_dependents FROM people WHERE id = 1;
=> #<People:0x007fa92b2adb00 @firstname="foo", @id=1, @lastname="bar", @number_of_dependents=
```

動く 😄

次は #update の実装

```
class People
  UPDATE_SQL = "UPDATE people SET lastname = :lastname, firstname = :firstname, number_of_dep

  def update
    self.class.con.execute(ActiveRecord::Base.send(:sanitize_sql_array, [UPDATE_SQL, attribut
  end

  def attributes
    { id: id, firstname: firstname, lastname: lastname, number_of_dependents: number_of_depen
  end
end
```

クエリを組み立てて実行するだけ。実行してみると

```
(main)> people = People.find(1)
(0.2ms)  SELECT id, lastname, firstname, number_of_dependents FROM people WHERE id = 1;
=> #<People:0x007fa92a47efc0 @firstname="foo", @id=1, @lastname="bar", @number_of_dependents=
// lastnameを書き換え
(main)> people.lastname = "hoge"
=> "hoge"

// update
(main)> people.update
(0.8ms)  UPDATE people SET lastname = 'hoge', firstname = 'foo', number_of_dependents = 100 W
=> []

// 取得し直してみるとlastnameが変わっているのがわかる
(main)> People.find(1)
(0.2ms)  SELECT id, lastname, firstname, number_of_dependents FROM people WHERE id = 1;
=> #<People:0x007fa92d82a7e8 @firstname="foo", @id=1, @lastname="hoge", @number_of_dependents
```

次は #insert の実装

```
class People
  INSERT_SQL = "INSERT INTO people(id, lastname, firstname, number_of_dependents) VALUES (:id

  def insert
    self.class.con.execute(ActiveRecord::Base.send(:sanitize_sql_array, [INSERT_SQL, attribut
  end
end
```

クエリを組み立てて実行するだけ。(保存済みかどうかみたいのは考慮してない)

実行してみると

```
// newでインスタンスを作成
(main)> people = People.new(id: 2, firstname: "hoge", lastname: "huga", number_of_dependents:
=> #<People:0x007fa92b3d6590 @firstname="hoge", @id=2, @lastname="huga", @number_of_dependent

// insert
(main)> people.insert
 (9.0ms)  INSERT INTO people(id, lastname, firstname, number_of_dependents) VALUES (2, 'huga'
=> []

// チートして用意した .all で確認すると、レコードが増えていることがわかる
(main)> People.all
People Load (0.2ms)  SELECT "people".* FROM "people"
=> [#<People:0x007fa92a6ab618 id: 1, lastname: "hoge", firstname: "foo", number_of_dependents
 #<People:0x007fa92a698018 id: 2, lastname: "huga", firstname: "hoge", number_of_dependents:
```

ビジネスロジックを実装してみる

```
class People
  def exemption
    base = 1500
    dependent = 750
    base + dependent * number_of_dependents
  end
end
```

本のサンプルがよく分かんなかったので、適当なメソッドを用意しました

実行してみると

```
(main)> People.find(1).exemption
(0.2ms)  SELECT id, lastname, firstname, number_of_dependents FROM people WHERE id = 1;
=> 76500
(main)> People.find(2).exemption
(0.1ms)  SELECT id, lastname, firstname, number_of_dependents FROM people WHERE id = 2;
=> 151500
```

終わり

案外簡単にさくっとActiveRecordパターンを実装出来た 🎉
(クエリビルダーとかコネクション周りでrailsのActiveRecord使ったけど。。)

使ったコード

```
require "active_record"

ActiveRecord::Base.logger = Logger.new(STDOUT)
ActiveRecord::Base.establish_connection(
  adapter: "sqlite3",
  database: "demo.sqlite3"
)

class Registory
  class << self
    def person(id) ;end
    def add_person(person); end
  end
end

class People
  FIND_SQL = "SELECT id, lastname, firstname, number_of_dependents FROM people WHERE id = ?;"
  UPDATE_SQL = "UPDATE people SET lastname = :lastname, firstname = :firstname, number_of_dep
  INSERT_SQL = "INSERT INTO people(id, lastname, firstname, number_of_dependents) VALUES (:id

  class << self
    # キャッシュ周りのロジックは省略しました
    def find(id)
```