ActiveRecord について

Release:2016/11(var2.0.0)

ActiveRecord 概要

ActiveRecord とはなにか

Ruby on Rails の OR マッピングのライブラリ。O/R マッピングは、「オブジェクト」と「リレーショナルデータベースの情報」を対応付けするための技術であり、データベースを利用するためのオブジェクトとデータベースとの間の橋渡しをする。
ActiveRecode 自体は Ruby on Rails を通さなくとも使用することが可能。

ActiveRecord の特徴

- テーブル対応
  1 つのクラスが 基本的に DB の 1 テーブルに対応する。
  クラスの属性は、テーブルの各カラムに対応します。

- クラス関係
  ActiveRecord::Base の派生クラスとして実装し、app/models 配下に格納する。

- オブジェクトと DB の対応
  モデルクラスの 1 インスタンス（オブジェクト）が、データベースの 1 レコードに対応する。
  オブジェクトが保持する属性の値が、レコードの保持する各カラムの値と対応する。

- オブジェクト指向
  モデルクラスは、そのモデルの表すロジックを追加できるので、オブジェクト指向的なコードで処理できる。

OR マッピングの機能

- コネクション管理機能
  データベースとの接続やコネクションプーリングが管理できるようになる。

- 自動マッピング機能
  XML ファイルなどの外部ファイルにオブジェクトの属性名とテーブルの列名に関するマッピング定義を設定することによって自動的にマッピング処理を行う。

- マッピングファイル、DTO、DAO の自動生成機能
  O/R マッピングツールの提供する自動生成ツールによってマッピングファイル、DTO、DAO といったファイルを自動作成。

- 接続情報の管理機能
  JDBC ドライバやデータベースの接続情報を、O/R マッピングツール側で管理。

- キャッシュ機能
  一度取得した検索結果をメモリ上に保持しておいて、同じ検索処理が行われた場合は、メモリ上にある検索結果を返す。データベースへのアクセスを減らし、パフォーマンスがあがる。

インストール

- SQLite のインストール

```
gem install sqlite3
```

- ActiveRecord インストール

```
gem install activerecord
```

ActiveRecord 基本構文

Main.rb ファイルを用意し ActiveRecord を使用する準備を行う

```
require 'active_record'

 ActiveRecord::Base.establish_connection(
        "adapter" => "sqlite3",
        "database" => "./[データベース名]"
 )
```

※require で active_record を使えるようにする

ActiveRecord を継承するクラスの宣言をする

```
class [クラス名] < ActiveRecord::Base
```

```
end
```

Create 構文（テーブル作成）

※SQL で直接テーブルを作成する

基本構文：

```
CREATE TABLE テーブル名（カラム名 1, カラム名 2, ...);
```

・カラム制約条件

NOT NULL [CONFLICT 句] |
PRIMARY KEY［ソート順] [CONFLICT 句] |
UNIQUE [CONFLICT 句] |
CHECK (評価式) [CONFLICT 句] |
DEFAULT 値

・制約条件

PRIMARY KEY (名前 [, 名前]*) [CONFLICT 句]|
UNIQUE (名前 [, 名前]*) [CONFLICT 句] |
CHECK (評価式) [CONFLICT 句]

・カラム制約

PRIMARY KEY	主キーを設定
NOT NULL	カラムの NULL 許可しない
UNIQUE	カラムの値がテーブル内で重複しない
CHECK	カラムの値を式により評価
DEFAULT	カラムのデフォルト値を指定

例）**アカウント管理テーブル**

```
CREATE TABLE accounts (
    id          integer NOT NULL,
    flag        integer DEFAULT 1,
    name        varchar(64) NOT NULL,
    email       varchar(128) UNIQUE,
    password    varchar(32),
```

```
    created_at,
    updated_at,
    CONSTRAINT account_pky PRIMARY KEY (id),
    CONSTRAINT password_chk CHECK (length(password) >= 5)
);
```

email：が重複して登録されないようにします。
password：4 文字以上入力必須

データ挿入

基本構文：

```
hoge = [クラス名].new(:[column] => "[value]")
hoge.save
```

```
hoge = [クラス名].new
hoge.[column] = "[value]"
hoge.save
```

```
[クラス名].create(:[column] => "[value]")
```

データ参照

基本構文：

全てのレコードを取得

```
[クラス名].all
```

最初のレコードを取得

```
[クラス名].first
```

最後のレコードを取得

```
[クラス名].last
```

インデックス番号でレコードを取得

```
[クラス名].find([index])
```

カラム名でレコードを取得※１つ

```
[クラス名].find_by([column]: "[value]")
```

Where 句構文：

Where (AND)

```
[クラス名].where(:[column1] => "[value1]" , :[column2] => "[value2]")
```

```
[クラス名].where("[column1] = ? and [column2] = ?", "[value1]" ,[value2])
```

```
[クラス名].where("[column1] = :[value1] and id = :id" ,{:[column1] => "[value1]" , :[column2] => "[value2]"})
```

Where (OR)

```
[クラス名].where("[column1] = ? or [column2] = ?", "[value1]" ,[value2])
```

```
[クラス名].where("[column1] = :[value1] or id = :id" ,{:[column1] => "[value1]" , :[column2] => "[value2]"})
```

Where (LIKE)

```
[クラス名].where("[column] like ?", "[like 形式]")
```

[like 形式]の例)

“test%” ※test の文字列の後に 0 文字以上の文字があるもの

Where (範囲指定)

```
[クラス名].where(:[column] => [範囲指定])
```

[範囲指定]の例)

1..5 は 1 から 5 までの範囲

[1,5]は 1 と 5 の指定

Orderby 句構文：

```
[クラス名].order("[column] [asc | desc]")
```

※昇順：asc　降順：desc

limit 句構文：

```
[クラス名].limit([個数])
```

scope 指定(limit の別名指定)

```
class [クラス名] < ActiveRecord::Base
        scope :[name], limit([個数])
end

[クラス名].name
```

データ更新

基本構文：

レコード更新　※１カラム

```
hoge = [クラス名].find([index])
hoge.update_attribute(:[column], "[value]")
hoge.save
```

レコード更新　※複数カラム

```
hoge = [クラス名].find([index])
hoge.update_attributes(:[column1] => "[value1]",:[column2] => "[value2]")
hoge.save
```

レコード更新　※複数レコード/ 複数カラム

```
[クラス名].where([条件]).update_all(:[column1] => "[value1]",:[column2] => "[value2]")
```

データ削除

基本構文：

```
hoge = [クラス名].find([index])
hoge.delete
```

```
hoge.save
```

レコード削除　※複数レコード/ 複数カラム

```
[クラス名].where([条件]).destroy_all
```