STI を避けたい

I wish I evacuate STI

yancya

RubyHiroba 2014
2014/09/21

自己紹介

- @yancya

STI って使ってます？

- どうですか？

STI とは

- Single Table Inheritance
- 単一テーブル継承
- モデルクラスを継承で表現し、永続化部分はスーパークラスのテーブル１枚でまかなう

STI の実装例（親クラス）

```ruby
class CreateCars < ActiveRecord::Migration
  def change
    create_table :cars do |t|
      t.integer :weight
      t.string :color
      t.string :type
      t.timestamps
    end
  end
end

class Car < ActiveRecord::Base
end
```

STI の実装例（親クラス）

```ruby
class CreateCars < ActiveRecord::Migration
  def change
    create_table :cars do |t|
      t.integer :weight
      t.string :color
      t.string :type
      t.timestamps
    end
  end
end

class Car < ActiveRecord::Base
end
```

STI の実装例（子クラス）

```ruby
class CreateManualCars < ActiveRecord::Migration
  def change
    # スーパークラスのテーブルにカラムを追加するだけ
    add_column :cars, :number_of_gears, :integer
  end
end

class ManualCar < Car
end

class AutomaticCar < Car
end
```

STI の便利さ

- サブクラス特有のカラムを add_column して、スーパークラスを継承するだけで作れちゃう便利
- スーパークラスに共通処理や属性を持たせることで、サブクラスのコードがスッキリ！

STI すごい

- メタデータカラムがあるので、親クラスのインスタンスから、子クラスを特定出来たりする

```
AutomaticCar.create(
  weight: 1000,
  color:  "blue"
) #=> #<AutomaticCar id: 1>

Car.find(1).tap{|car|
  break car.type.classify.constantize.find(car.id)
}.class #=> AutomaticCar
```

STI の注意点

```ruby
# スーパークラスからサブクラスの属性に触れちゃう
Car.find(1).number_of_gears #=> 6
Car.find(1).update(number_of_gears: 5)

# それを防ぐために、守るコードを追加
# gem 'protected_attributes'
class Car < ActiveRecord::Base
  attr_accessible :weight, :color
end

class ManualCar < Car
  attr_accessible :number_of_gears
end
```

外部キー制約

- ついに Rails 本体に外部キー制約サポートがくるよー

> *Support for real foreign keys!*
>
> *add_foreign_key/remove_foreign_key are now available in migrations.*

[「http://weblog.rubyonrails.org/2014/8/20/Rails-4-2-beta1/」より引用]

外部キー制約って使います？

- みなさん、どうですか？

STI と外部キー制約

- 外部キー制約を使うとして
- 子クラス特有の属性を定義
- その属性が外部キー
- その属性に NOT NULL 制約を付けたいとする

STI で外部キー制約

- 親や他の子クラスから INSERT したら NULL が入っちゃう（入らなくてエラー）
- いいこと思いついた！外部テーブル側に {id: 99999, value: "無し"} みたいなレコードを入れておいて、外部キーに NULL が入りそうになったら、代わりに 99999 を

例えば、STI を避ける

- STI を避ける必要は全然無い
- 呪われし RDB 厨の業
- Nullable column
- 正規化
- では、しかるべきデータ格納方法とは

STI の背景

> *Relational databases don't support inheritance, so when mapping...*

[「http://www.martinfowler.com/eaaCatalog/singleTableInheritance.html」より引用]

RDBMS

- みなさん、何使ってます？

PostgreSQL

- RDBMS 界の優等生
- バージョン毎の日本語ドキュメントが充実
- Heroku のデフォルト RDBMS
- v9.2 から JSON 型をサポート（v9.3 で更に高機能に）

Table Inherited

- "Relational databases don't support inheritance, so when mapping..."
- テーブルの継承 PostgreSQL にはあるんです

継承先テーブルの作成

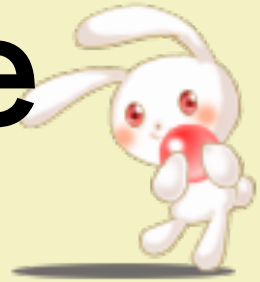
```ruby
class AddManualCar < ActiveRecord::Migration
  def up
    execute <<-SQL
      CREATE TABLE manual_cars(number_of_gears integer)
          INHERITS cars
    SQL
  end

  def down
    drop_table :manual_cars
  end
end

class AddAutomaticCar < ActiveRecord::Migration
  def up
    execute "CREATE TABLE automatic_cars() INHERITS cars"
  end

  def down
    drop_table :automatic_cars
  end
end
```

Where a row come from?



> *この列は特に、継承階層からの選択問い合わせでは便利です。tableoidはテーブル名を得るためにpg_classのoid列に結合することができます。*
>
> *[「システム行 - tableoid」より引用]*

Behavior of Parental Model

```ruby
module Parental
  def down_cast
    sub_model.find(self)
  end

  private
  def sub_model
    relation_name = ActiveRecord::Base.connection.execute(<<-SQL).first["relname"]
      SELECT relname
        FROM #{self.class.to_s.tableize} p, pg_class c
       WHERE p.tableoid = c.oid
         AND p.id = #{self.id}
       LIMIT 1
    SQL

    relation_name.classify.constantize
  end
end
```

Behavior of Inherit Model

- 継承したままだと table_name がスーパークラスのもの

```
module PgInherits
  extend ActiveSupport::Concern

  included do |base|
    base.table_name = base.name.tableize
  end
end
```

Model Classes

```ruby
class Car < ActiveRecord::Base
  include Parental
end

class ManualCar < Car
  include PgInherits
end

class AutomaticCar < Car
  include PgInherits
end
```

テーブル継承の注意点

- 多重継承ができる
- 単にデータベースの機能として見れば便利
- 多重継承の無いプログラミング言語とマッピングしづらい

テーブル継承の注意点

- 親テーブルで PRIMARY KEY, UNIQUE の宣言をしていても、子テーブルまでは制約が伝播しません
- 親テーブルに外部キーを持って、外部キー制約をつけていても、子テーブルまで制約が伝播しません

テーブル継承の注意点

- NOT NULL
- DEFAULT
- これらは子テーブルに伝播します

何が起こりうるか

- PRIMARY KEY, UNIQUE について、親まで遡らないので、子テーブルから重複値を入れ放題
- 子テーブルに別途制約を付けても、子テーブル内でしかチェックしないので根本解決にならない

具体的に困ること

```
Car.create #=> #<Car id: 1>
AutomaticCar.create(id: 1) #=> #<AutomaticCar id: 1>
Car.where(id: 1).count #=> 2
```

ただ、DEFAULT は伝播するので、ID シーケンスは親テーブルと共通なので、ID を直接指定して INSERT UPDATE しなければ問題ない。しなければ

回避策

```
module PgInherits
  extend ActiveSupport::Concern

  included do |base|
    base.table_name = base.name.tableize
    # gem 'protected_attributes'
    attr_protected :id
  end
end
```

テーブル継承のこれから

- PostgreSQL 9.3.2文書 5.8.1 によれば
- 「これらの機能の不足は、今後のリリースでおそらく改善されるでしょう」とのこと
- ただ、まぁ Version 7.4 からずっと書いてあるので

まとめ

- おとなしく STI を使いましょう
- でも、必要がなくなったら使うのをやめたい

質疑応答

- もし質問がなければ、こちらから皆さんに質問したり
- まめ知識を披露しますね