Ładowanie kodu w Rails

Czyli co zrobić, by trzymać kilka klas w jednym pliku

```ruby
# app/models/blog.rb

class Post
end

class Comment
end

# app/controllers/blog.rb

class PostsController < ApplicationControlle
end

class CommentsController < ApplicationContro
end
```

Oczekujemy, że w trybie development (cache_classes = false) klasy będą się przeładowywać razem ze stroną

Plan

mechanizmy języka Ruby, które są potrzebne

ActiveSupport::Dependencies

3 rozwiązania problemu kilku klas (dobre, takie sobie i całkiem złe)

Mechanizmy Ruby

load "plik.rb"

require "plik"

kluczowa różnica: require nie wczyta tego samego pliku dwukrotnie

Wniosek

require się nie nadaje

autoload :Klasa, "plik"

kiedy bedzie potrzebna Klasa, wczytaj plik

autoload używa require

Wniosek

autoload się nie nadaje

Module#const_missing(name

kiedy brakuje jakiejś klasy, Ruby uruchamia metodę const_missing, dając jeszcze jedną szansę na załadowanie

Module#remove_const(name

usuwa klasę

ActiveSupport::Dependencie

1) odgaduje nazwę pliku z klasą i ładuje ją, kiedy jest potrzebna

2) potrafi usunąć wczytane klasy

1) odgaduje nazwę pliku z klasą i ładuje ją, kiedy jest potrzebna

nadpisuje const_missing

nazwa pliku zgodna z konwencją:
NazwaKlasy w nazwa_klasy.rb

config.autoload_paths, domyślnie zawiera app/models, app/controllers, itd.

2) potrafi usunąć wczytane klasy

new_constants_in { ... }

ActiveSupport::Dependencies.clear

require_dependency "plik"

A co w trybie produkcyjnym?

config.cache_classes = true wyłącza większość logiki i powoduje użycie require zamiast load

Sposób 1 (dobry)

Ładowanie modeli

```
# application_controller.rb
# lub dowolny inny kontroler

require_dependency "app/models/blog"
class ApplicationController < ...
```

Ładowanie kontrolerów

jest problem, bo ApplicationController też jest ładowany automatycznie, kiedy wykonuje się dziedziczenie

gdzie wrzucić require_dependency, może initializer?

```
# config/initializers/load_classes.rb

require_dependency "app/controllers/blog"
```

zadziała tylko przy pierwszym żądaniu :-(

Rozwiązanie: ActionDispatch::Callbacks

```
# config/initializers/load_classes.rb

ActionDispatch::Callbacks.before do
  require_dependency "app/controllers/blog"
  require_dependency "app/models/blog"
end
```

Sposób 2 (taki sobie)

nadpisać const_missing i udawać autoload, który używa require_dependency zamiast require

```ruby
require Rails.root.join "lib", "custom_const_missing"
CustomConstMissing.hook!

CustomConstMissing.map :Post, "app/models/blog"
CustomConstMissing.map :PostsController,
    "app/controllers/blog"
```

Sposób 3 (całkiem zły)

jest hackiem

w ogóle nie działa

require (a więc i autoload) korzysta z tablicy załadowanych plików $LOADED_FEATURES / $"

można usunąć coś z tej tablicy, żeby require załadował drugi raz

Przykład

```ruby
file_name = File.expand_path("../klasa.rb", __FILE__)

autoload :Klasa, file_name
Klasa # działa

Object.instance_eval { remove_const :Klasa } # usuń
Klasa rescue nil # nie działa, wyrzuca NameError

$LOADED_FEATURES.delete file_name
Klasa # znowu działa
```

nie udało mi się tego uruchomić w Rails

ale może się jakoś da :-)

o ile w ogóle warto

Koniec