Ruby on Rails
中級者を目指して

株式会社万葉　大場寧子

y.ohba@everyleaf.com

自己紹介

- 実装大好きプログラマ
- … → C++ → Java(7年) → Ruby(1.5年)
- Award on Rails 2006 大賞受賞

実装大好き

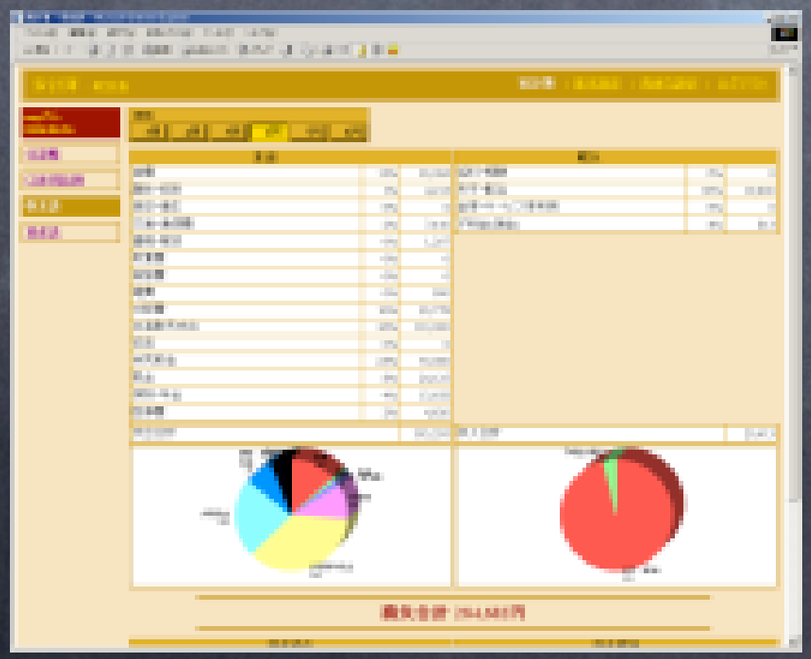
- ベンチャーでマネージャ
- ・・・・orz
- やっぱ実装がしたい

開発体制・手法

- アジャイル大好き
- テスト駆動　それなりに
- 習慣、マインド、コミュニケーション重要

Award on Rails 2006

- Web家計簿「小槌」
- 大賞
- まつもとゆきひろさん審査員特別賞



http://www.kozuchi.net

それから１年・・・

- Ruby on Rails の仕事100%
- ECサイト
- 医事会計システム
- SNS

現在の仕事

- Ruby on Rails による開発支援
- 会社を作った
  - 株式会社万葉
  - http://www.everyleaf.com

本日のテーマ

- Ruby on Rails で実践的にモノを作ろう
- 初級者向け講座/入門本の次ステップへ
- 高度な話題も少し

みなさんの状況は？

- Ruby on Rails を仕事で使ったことがある？
- Java 経験者？

Hello, RoR

- まずこんなことを勉強しますよね
- Scaffold
- データベースのCRUD
- Session, Request

開発を始めると

- とにかくコードを書いて目的を果たす
- あ、これ Rails に機能があったんだ
- あ、これもあったんだ
- 全然 Ruby らしくないコードになってた
- 最初からこれを使えばよかった etc...

ということで

- 結局は、作らないと身につかない
- でも、いいヒントがあれば効率的
- 日頃よく使うAPI、パターン
- 先に知っておきたかった Rails機能, Plugin
- Ruby らしいコードにするコツ

熟練へのキーワード

RESTな設計

Hash と Array

セキュリティ

ActiveRecord

Module による Mix-in

DRY

キャッシュ

Ajax

よくある問題と解決パターン

Plugin

よい土壌で育てよう

- ツールを揃える - IDE, SVN, Trac, Wiki, UML
- よい名前をつける
- テストを書く
- 設計してから実装する
- 愛

出発！

Java経験者のための Rubyのコツ

- 0も真なり
- ||=
- 1行で書いてみる
- Hash と Array
- クラスメソッド

Ruby - 0も真なり

```
s = 0
if s
  p "0も真なり"
end
```

- 偽は nil と false だけ
- 0 も真

nil チェック

```
if value != nil
   ...
```

かっこわるい

```
if value
   ...
```

通常これで十分

```
unless value.nil?
   ...
```

厳密なチェック

1行で書いてみる

```
if !id
  raise "no id"
end
```

Java から来ると１行に抵抗感

```
raise "no id" if !id
```

中身が１行なら１行が見やすい

```
raise "no id" unless id
```

異常判定は unless 向き

Hash と Array

- これがなくては始まらない
- APIをよく読んで何ができるか覚えよう

Hash

```
hash = {key => value, key => value, ...}
```

```
value = hash[key]
```

記法に目を慣らそう

```
obj.foo( {key => value, key => value} )
```

```
obj.foo(key => value, key => value)
```

引数のハッシュの{} は省略できる場合も

名前つき引数のように使われる

Array

自分でゴリゴリ走査するのはほとんどが無駄

```
admin= nil
for element in @array
  if element.name == "admin"
    admin = element
    break
  end
end
```

```
admin = @array.detect{|e| e.name == "admin" }
```

クラスに関する特色

- クラスメソッドも継承される
- リフレクションが強力。メソッドの追加、書き換えなどが簡単
- public, protected, private がJavaと微妙に違う
- 名前空間の解決は Module で

||=

```
name ||= "名無し"
```

- 偽だったら代入
- デフォルト値の挿入に便利
- 省略しないで書くと以下のようになる

```
name = name || "名無し"
```

ex) value += 1

private, protected, public

どこから呼ばれるか		インスタンス内			インスタンス外					
		クラス	派生クラス		クラス	派生クラス		他		
			パ内	外		パ内	外	パ内	外	
private	Ruby	○	○		×	×		×		
	Java	○	×	×	○	×	×	×	×	
protected	Ruby	○	○		○	○		×		
	Java	○	○	○	○	○	×	○	×	
なし	Java	○	○	×	○	○	×	○	×	
public	Ruby	○	○		○	○		○		
	Java	○	○	○	○	○	○	○	○	

Rubyは

- 防衛より自由を重視
  - 派生クラスに対してオープン
- Javaよりインスタンスの内外を区別する
  - レシーバ

レシーバ

```
item.save
```

item の部分がレシーバ

```
self.save
```

self だってレシーバ

```
save
```

レシーバなしのメソッド呼び出し

この形ならprivate メソッドも呼べる

RoR での使い方

- 公開メソッド - public
- それ以外 - private
- protected は以下の場合に使う
  - ActiveRecordのコールバックを直接書く時
  - インスタンスを超えた処理で完全公開はしたくないもの

クラスメソッド

```
class Foo
  def self.foo
    p 'foo'
  end
end
```

呼ぶときは

```
> Foo.foo
```

```
> f = Foo.new
> f.class.foo
```

本日のメインテーマ

ActiveRecord

ぜひ覚えてほしいこと

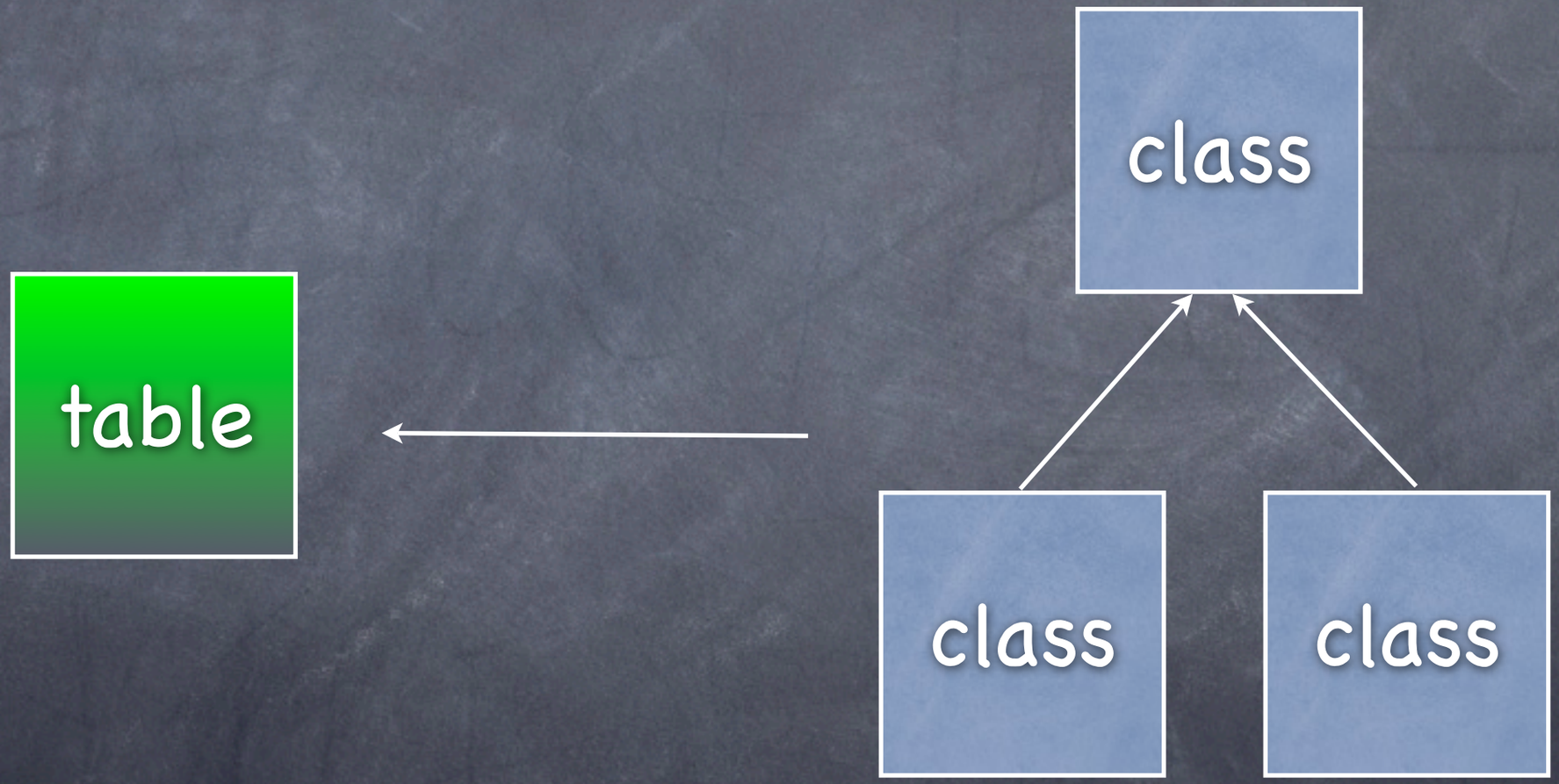
- 単一テーブル継承
- ポリモーフィック関連
- コールバック
- :joins と :include

その他の話題

- 検証
- カウンターキャッシュ
- Acts as List

単一テーブル継承

継承関係にある複数のクラスを



１つのテーブルに対応づける

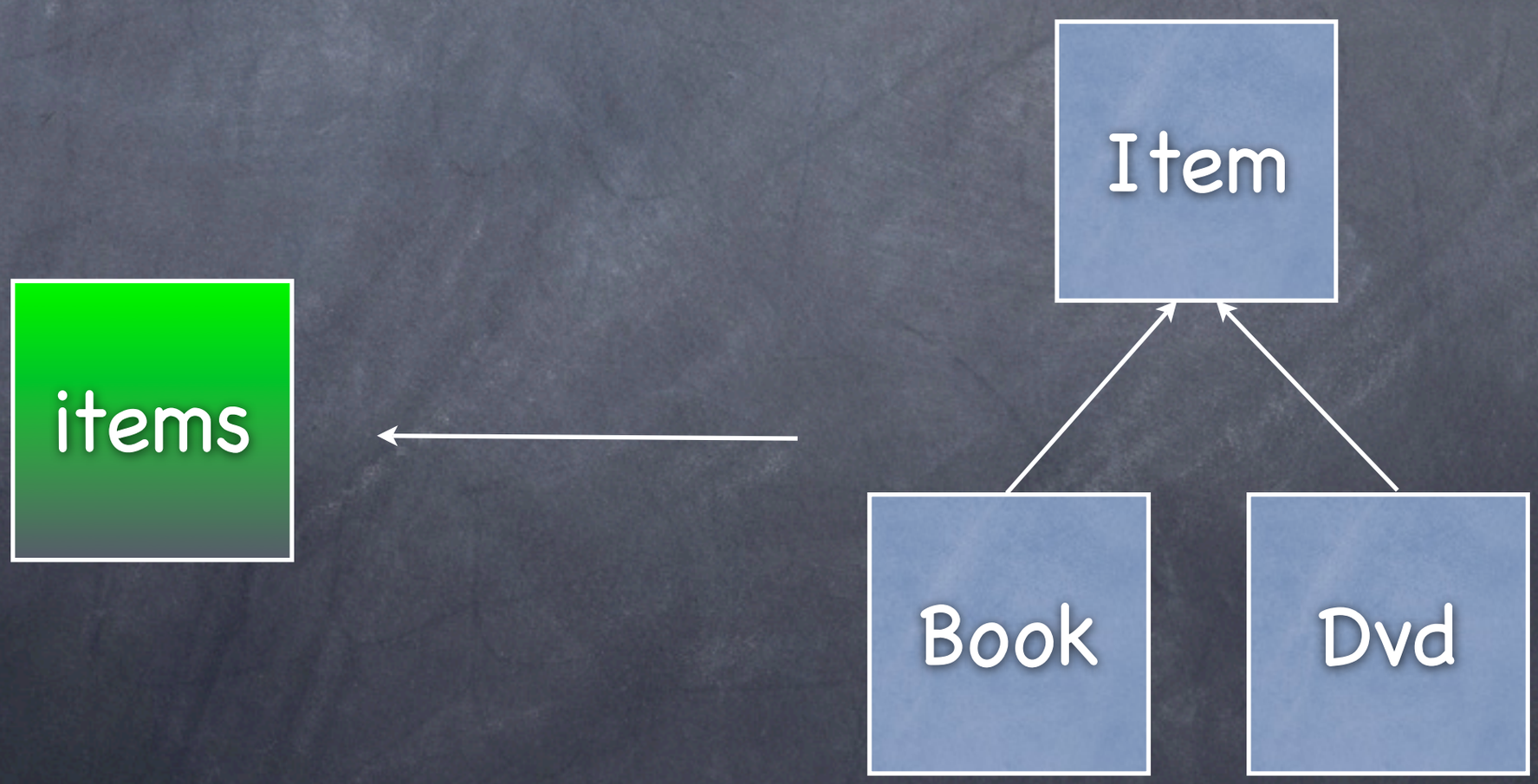
こんなとき使う

- 似ているけどちょっと違うモデル同士を継承を使って実装したい
- だいたい同じようなカラムを持っているので同じテーブルに格納したい
- 抽象化されたクラスのレベルでも処理したい

例）

Item, Book, Dvdクラスを

みんな items テーブルに格納



単一テーブル継承の発動条件

- テーブルに文字列を格納できる type カラムがあること
- 同じテーブルに入れたいクラス同士が継承関係でまとまっていること

type カラム

items テーブル

type: varchar(30)

"Book"

Item

Book

Dvd

type カラムにクラス名が格納される

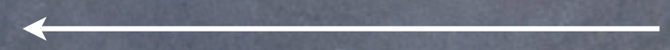
ポリモーフィック関連

- 複数のテーブルを１つの関連に抽象化して扱う
- 例
  - 事業拠点には工場と事務所があり、別のテーブルで管理している
  - まとめて扱いたい処理がある
  - 例えば、Employee belongs_to :work_place

関連のおさらい

has_many :employees

has_one :employee



employees

Employee

Office

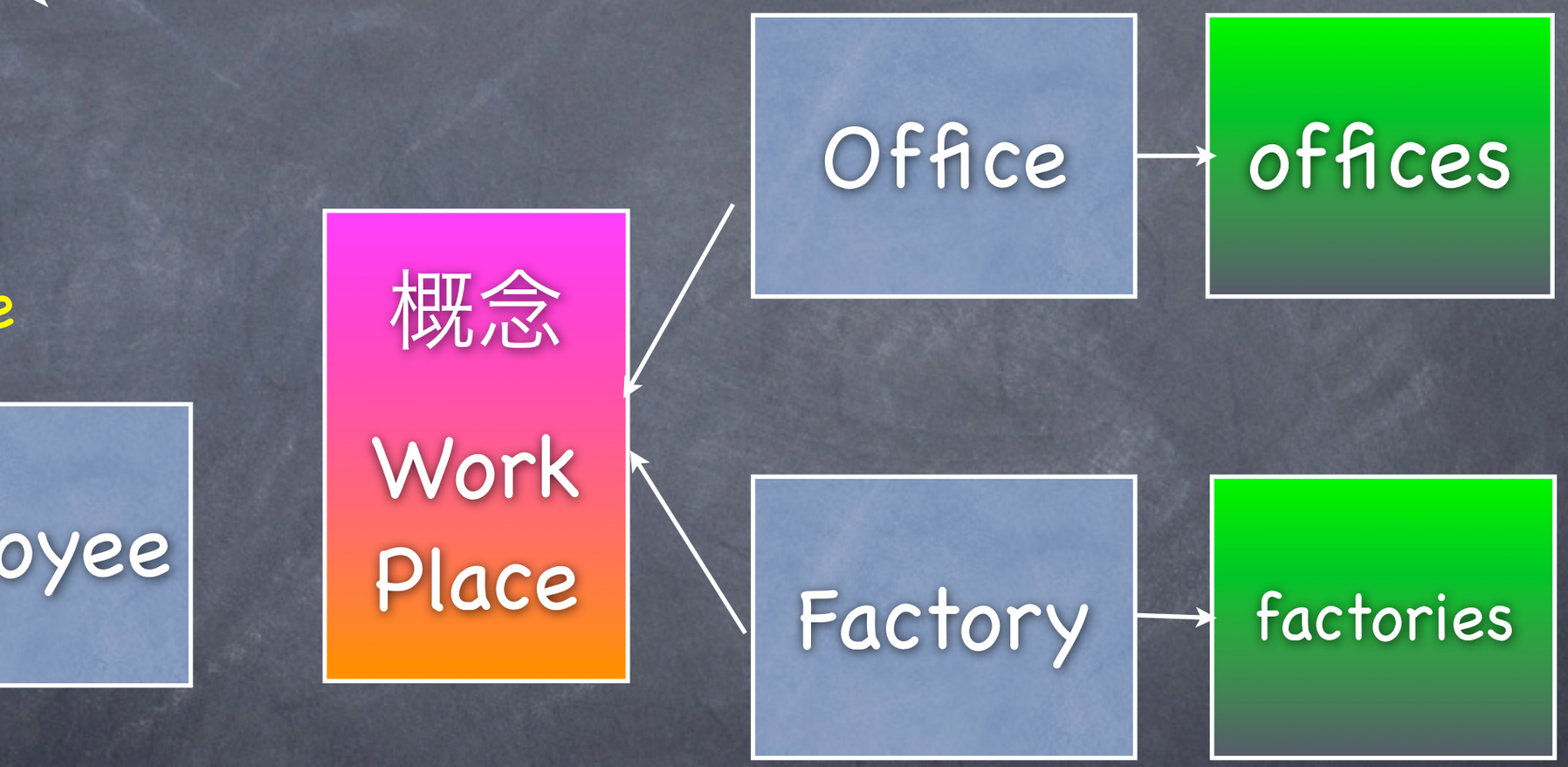
offices

office_id

belongs_to :office

ポリモーフィック関連

has_many :employees, :as => work_place



belongs_to :work_place, :polymorphic => true

work_place_id, work_place_type

- xxx_id, xxx_type の xxx は自由
- id と type(クラス名)がセットで格納される
- 通常、xxx_type は自動で条件に追加
- has_many :through を使った多対多など、ポリモーフィック関連と組み合わせて使えないフィーチャーもあるが何とかなる。

両者の類似点

- 単一テーブル継承とポリモーフィック関連は、クラス名を(xxx_)typeカラムに入れる
- データベース設計とクラス設計の差を吸収してくれる　→　クラス設計の自由度Up！
- 設計前に理解しておきたい

コールバック

- ActiveRecord モデルインスタンスのライフサイクルにあわせて呼ばれる
- インスタンス構築、Save、検証、削除
- ビジネスロジックを書くときは、コールバックに入れられないかまず考えるとよい

例えばこんな利用例

```
# 親を更新したら子も強制的に更新
def after_update
  children.each {|c| c.save!}
end
```

余談 - 上達の流れ

Controller にコードを書きまくる

↓

Modelに独自メソッドを書く

↓

Rails APIを使うことで解決

↓

コールバックへ移す

↓

正しい解だった

:joins と :include

特定の都道府県に勤めている社員を検索

```
employees = Employee.find(:all,
    :select => "employee.*",
    :conditions => ["offices.prefectue_id = ?",
                    prefecture_id],
    :joins => "left join offices on
               employees.office_id = offices.id")
```

:joins はSQL の join 部分

:joinsと:include (2)

社員オブジェクトのついでに

職場オブジェクトも取得

```
class Employee < ActiveRecord::Base
  belongs_to :office
end
```

```
employee = Employee.find(@id,
                :include => :office)
```

検証

- Rails に用意されたよくある検証API
  - validates_presence_of etc...
  - 自分でも書ける
  - 検証 = errors にエラーを登録する

```
def validate
  errors.add_to_base("奇数はだめ！") unless quantity % 2 == 0
end
```

検証に関するポイント

- 大部分は既存のAPIで足りる
- :scope, :allow_nil など細かい指定ができる
- エラーメッセージがデフォルトだと英語！
  - 以前は ActiveHeart Plugin
  - 最近は GetText , Scpecial Generator など

Acts as List

- has_many で定義した子モデルたちに「順序」がある
- 例）
  - 家計簿の同じ日の記入
  - 複数著者がいる場合の並び方

Acts as List 実装

- テーブルに position カラムを用意
- モデルに以下を記述

```
class Note < ActiveRecord::Base
  has_many :lines, :order => "position"
end
```

```
class Line < ActiveRecord::Base
  belongs_to :note
  acts_as_list :scope => :note
end
```

rails 2.0 では
プラグインへ

カウンターキャッシュ

- has_many 関係にある子モデルが何個あるかを、親モデルのカラムに格納しておく
- 編集にくらべて集計結果の表示が頻繁な要件では高速に処理できて便利
- 例) 事務所一覧で、所属社員数を表示したい

カウンターキャッシュの例

```
class Employee < ActiveRecord::Base
  belongs_to :office, :counter_cache => true
end
```

★offices テーブルに、employees_count というカラムを用意

★デフォルトを0にする

ActiveRecord編 終了

- API仕様書を何度でもよく読みましょう
- 特にAssociation 系 (has_manyなどがのってる)

http://api.rubyonrails.org/

RESTな設計

- Controller 設計はまず URL 設計から
- 記事にコメントをつけるアクションとして以下の
  どっちがかっこいい？
- entries/new_comment/1
- entries/1/comments/new
- 後者なら entries/1/comments/3 などに発展

RESTな設計は経済的

- 操作対象のモデルクラスごとにController
- 短い統一されたメソッド名
- routes.rb が定義しやすい
- Filter が機能しやすい
  - 親オブジェクト（先の例では@book）の存在チェックを一斉にできて見やすい

rails 2.0

- このあたり（RESTな設計）に関するサポートが入るらしい

よくある問題とヒント

- 認証
- 権限
- 親子モデルをセットにした編集
- 論理削除
- アップロード・画像処理
- ローカライゼーション（多言語対応）

認証

- acts_as_authenticated plugin 他
- before_filter を利用
- 個人的によくやる手法

```
before_filter :load_user
private
def load_user
  @user = User.find(session[:user_id])
end
```

権限

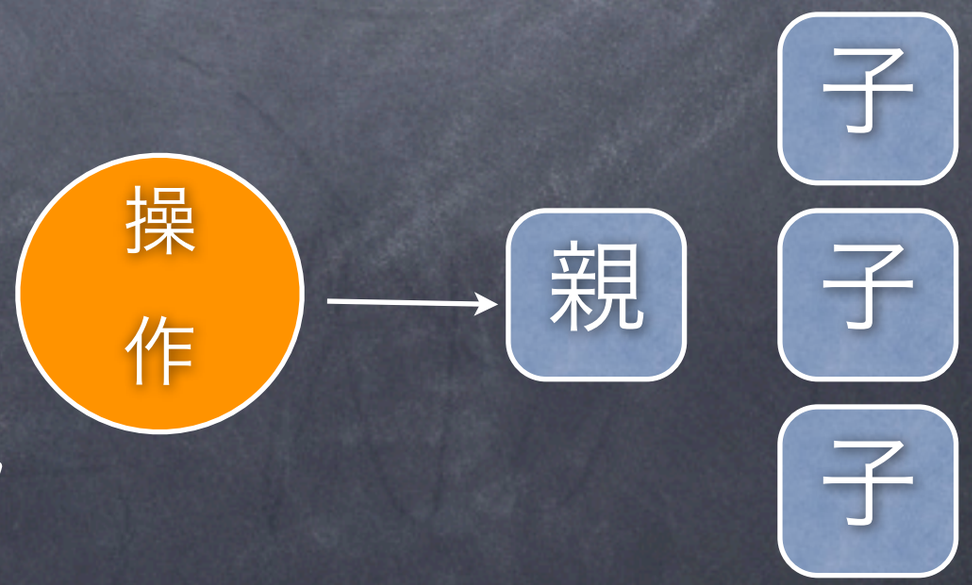
- 今のところ、要件ごとに作ることにしている
- 個人的にはモデルに管理させる方法をよく使う

```
@entries = Entry.secure_find(@user, options)
```

ユーザー（要求者）を渡し、権限のあるものだけをもらう。
返ってきたオブジェクトはどんなアクションが許可されているかしっており、違反すると例外。

親子モデルをセットにした編集

- 全部 create するときは親のsaveで保存される
- update のときは子は自動では更新されない
- 子の削除を親更新と同期させたいときは自前でケアが必要
- validates_association
- :dependent => :destroy



論理削除

- acts_as_paranoid プラグイン
- 簡単
- 削除、検索、関連など、各方面を拡張

画像アップロード

- file_column プラグイン
- RMagickと一緒につかう
- 手軽にアップロードと画像処理ができる
- I/F・モデルの柔軟性に凝った別のプラグインを公開予定だったりします

多言語対応

- GetText (gem で入れる
  パッケージ)
- Globalize プラグイン

使ったことのある
その他のプラグイン

- acts_as_taggable
  - タグづけ
- SpecialGenerator
  - 高機能な Scaffold

プラグインとのつきあい

- 要件に合うものを使う。無理に使わない。
- ソースを読む必要は出てくるものです
- プラグインを読むのは勉強になる
- Rails のバージョンアップに注意

RubyをDSLとして使う

DSL = ドメイン特化言語

```
has_many :employees
```

- ARのクラスメソッドを呼んでいるとは通常、意識しない
- Rails 専用記法のように理解される
- = DSL として使っている

黒魔術っぽい領域

- 例）String クラスに独自のメソッドが追加されている
  - under_score, pluralize ...
- acts_as_paranoid プラグインをいれたらなぜ論理削除ができるようになるの？
- どうやっているの？

ヒント

- クラスにメソッドを追加する
- Module による Mix-in
- メタプログラミング

クラスにメソッドを足す

- Java ではロードしてからメソッドを足すのは大変
- Ruby だと以下を実行するとできちゃう

```
class Object
  def yeah
    p 'yeah!'
  end
end
```

Module の Mix-in

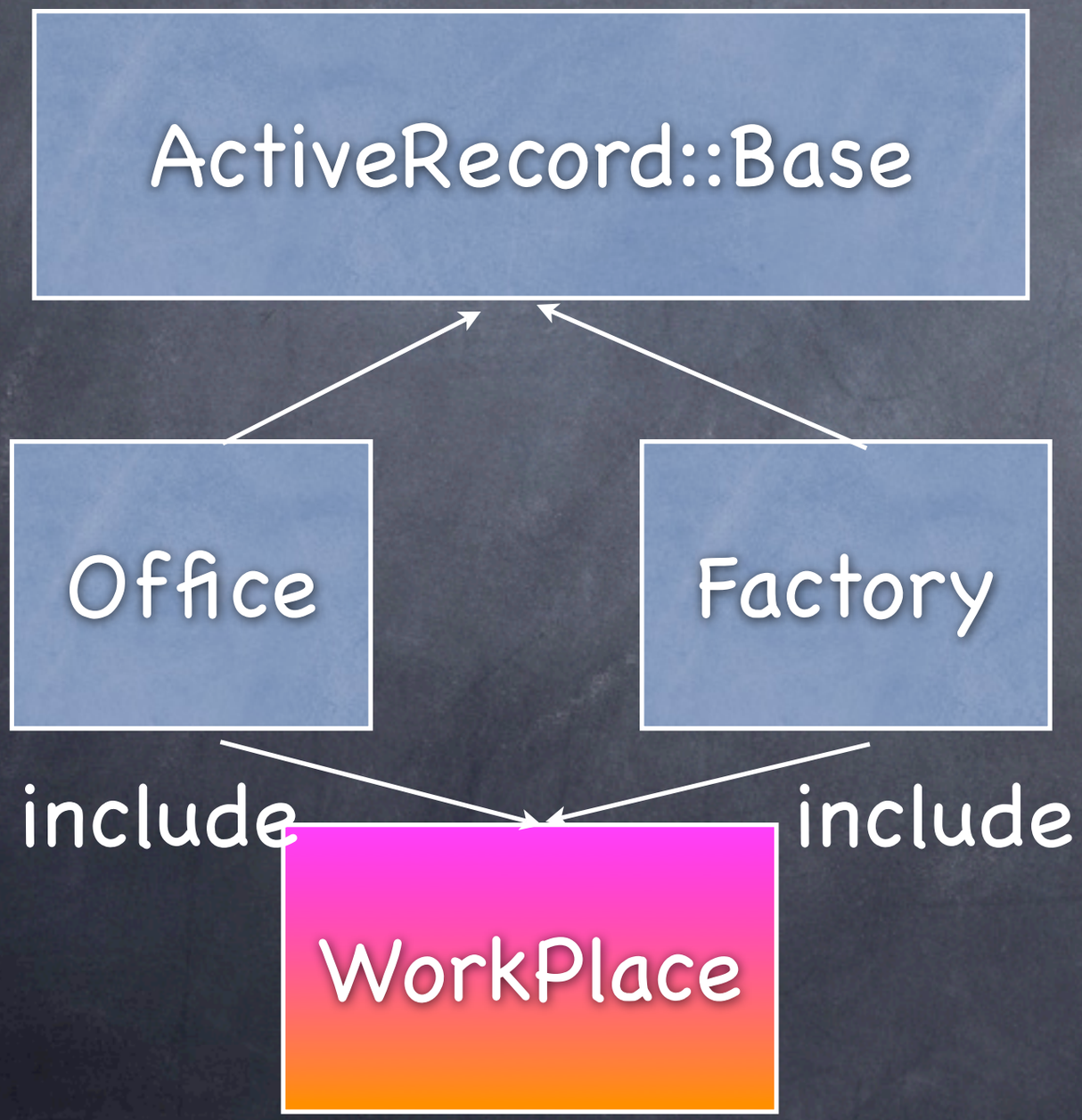
- Module を使ってインスタンスメソッドをクラスに持たせることができる

```
module Human
  def age(date)
    # 誕生日から計算(略)
  end
end
```

include

```
class Employee < ActiveRecord::Base
  include Human
end
```

多重継承/アスペクト指向っぽく



継承関係がつかえなくても

Moduleを使って

共通処理を切り出せる

include させて拡張する

```
module WorkPlace
  # 職場モデルはどれも社員とつながってる
  def self.included(base)
    base.has_many :employees,
                  :as => :work_place
  end
end
```

クラスメソッドも Moduleから渡せる

```
module WorkPlace
  module ClassMethods
    def foo
      p 'foo'
    end
  end
  def self.included(base)
    base.extend(ClassMethods)
  end
end
```

private メソッドを
どうしても呼びたい？

```
obj.__send__(:himitsu)
```

※Ruby1.9では別の方法をとることに

メソッドを動的に足す

```
clazz.___send___(:define_method, :yeah) {
  p `yeah!`
}
```

削除、再定義などもできる

やりたい放題 (^o^)/

プラグインの init.rb

- アプリケーション実行前に読み込まれる
- 既存のクラスを静的に拡張
- Module を読み込ませ、既存のクラスに include させ、クラスメソッドを追加
- → DSLっぽく使えるようになる
- 例) acts_as_paranoid

挑戦しましょう

- Module や Helper を活用してコードの重複をなくす
- DSLっぽく作っていくとかっこいい
- 将来の生産性がどんどんあがっていく
- まずは重複したコードをどうにかしたいという感性・情熱・執念から

本日の裏テーマ

男女仲良く
エンジニアリング
あるいはモテについて

注）この先の話は、あくまでも自分が
傾向として感じることを述べるだけで
あり、事実ではありません。
何も決めつけるつもりはありません。

エンジニアリングに関して
感じている男女の傾向の違い

- 男性
  - 原理・原則が好き
  - なぜそうなるかを深く理解したい
  - エンジニアリングの歴史・蘊蓄が好き

続き

- 女性
  - 現象・現実に最大の関心がある
  - 問題解決　＞　原理の理解
  - たくさんの What を解決したい
  - 共感を得たい

続きは懇親会で(^o^)/

楽観的予測

- 応用＝アプリケーション開発には、基盤系よりも多くの女性が関心を示すはず
- Ruby on Rails では下位レイヤーの処理はフレームワークが吸収。基盤的世界への耐性が低くても活躍できる度合いがアップ
- 女性エンジニアは増えるはず（@ｗ@

本日の裏テーマ

その２

Award on rails 2007

- 蔵書管理システム「BookScope」
- http://ko.meadowy.net/bookscope/
- 久保さんの手伝い

BookScopeの特徴

- Amazon WebServiceを利用してデータ取得
- バーコードリーダーで登録
- 本棚.org インポート
- フィード配信
- 蔵書の貸し借りを管理できる

一般投票

明日 10/19(金) 正午まで !!

http://rails.drecom.jp/vote

気に入っていただけましたら

BookScope に

清き一票をよろしくお願いします

m(_ _)m

ご静聴

ありがとうございました。