**EBook Gratuito**

# APPRENDIMENTO
# Ruby on Rails

Free unaffiliated eBook created from
**Stack Overflow contributors.**

#ruby-on-rails

# Sommario

# Di

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: ruby-on-rails

It is an unofficial and free Ruby on Rails ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official Ruby on Rails.



Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

# Capitolo 1: Iniziare con Ruby on Rails

## Osservazioni


Ruby on Rails (RoR), o Rails, è un framework di applicazioni web popolare open source. Rails utilizza Ruby, HTML, CSS e JavaScript per creare un'applicazione Web che gira su un server web. Rails utilizza il pattern model-view-controller (MVC) e fornisce una serie completa di librerie dal database fino alla vista.

## Versioni

| Versione | Data di rilascio |
|---|---|
| 5.1.2 | 2017/06/26 |
| 5.0 | 2016/06/30 |
| 4.2 | 2014/12/19 |
| 4.1 | 2014/04/08 |
| 4.0 | 2013/06/25 |
| 3.2 | 2012-01-20 |
| 3.1 | 2011-08-31 |
| 3.0 | 2010-08-29 |
| 2.3 | 2009-03-16 |
| 2.0 | 2007-12-07 |
| 1.2 | 2007-01-19 |
| 1.1 | 2006-03-28 |

| Versione | Data di rilascio |
| --- | --- |
| 1.0 | 2005-12-13 |

# Examples

## Creazione di un'applicazione Ruby on Rails

Questo esempio presuppone che *Ruby* e *Ruby on Rails* siano già stati installati correttamente. Altrimenti, puoi trovare come farlo **qui** .

Apri una riga di comando o un terminale. Per generare una nuova applicazione di rotaie, usa il comando rails new seguito dal nome della tua applicazione:

```
$ rails new my_app
```

Se si desidera creare l'applicazione Rails con una versione specifica di Rails, è possibile specificarla al momento della generazione dell'applicazione. Per farlo, usa le `rails _version_ new` seguito dal nome dell'applicazione:

```
$ rails _4.2.0_ new my_app
```

Ciò creerà un'applicazione Rails chiamata `MyApp` in una directory `my_app` e installerà le dipendenze gemme già menzionate in `Gemfile` utilizzando l' `bundle install` .

Per passare alla directory dell'app appena creata, utilizzare il comando `cd` , che rappresenta la `change directory` .

```
$ cd my_app
```

La directory `my_app` ha un numero di file e cartelle generati automaticamente che costituiscono la struttura di un'applicazione Rails. Di seguito è riportato un elenco di file e cartelle che vengono creati per impostazione predefinita:

| Cartella di file | Scopo |
| --- | --- |
| app / | Contiene controller, modelli, viste, helper, mailer e risorse per la tua applicazione. |
| bidone/ | Contiene lo script rails che avvia la tua app e può contenere altri script che utilizzi per impostare, aggiornare, distribuire o eseguire la tua applicazione. |
| config / | Configura percorsi, database e altro della tua applicazione. |
| config.ru | Configurazione rack per server basati su rack utilizzati per avviare l'applicazione. |
| db / | Contiene lo schema del database corrente e le migrazioni del database. |

| Cartella di file | Scopo |
|---|---|
| Gemfile<br>Gemfile.lock | Questi file ti permettono di specificare quali dipendenze gem sono necessarie per la tua applicazione Rails. Questi file sono usati dalla gemma di Bundler. |
| lib / | Moduli estesi per la tua applicazione. |
| log / | File di registro dell'applicazione. |
| pubblico/ | L'unica cartella vista dal mondo così com'è. Contiene file statici e risorse compilate. |
| Rakefile | Questo file individua e carica attività che possono essere eseguite dalla riga di comando. Le definizioni delle attività sono definite in tutti i componenti di Rails. |
| README.md | Questo è un breve manuale di istruzioni per la tua applicazione. Dovresti modificare questo file per dire agli altri cosa fa la tua applicazione, come configurarla ecc |
| test/ | Test unitari, infissi e altri apparati di prova. |
| temp / | File temporanei (come i file cache e pid). |
| vendor / | Un posto per tutto il codice di terze parti. In una tipica applicazione Rails questo include gemme vendute. |

Ora devi creare un database dal tuo file `database.yml` :

5.0

```
rake db:create
# OR
rails db:create
```

5.0

```
rake db:create
```

Ora che abbiamo creato il database, dobbiamo eseguire le migrazioni per configurare le tabelle:

5.0

```
rake db:migrate
# OR
rails db:migrate
```

5.0

```
rake db:migrate
```

Per avviare l'applicazione, è necessario attivare il server:

```
$ rails server
# OR
$ rails s
```

Per impostazione predefinita, i binari avvieranno l'applicazione alla porta 3000. Per avviare l'applicazione con un numero di porta diverso, dobbiamo accendere il server come,

```
$ rails s -p 3010
```

Se navighi su http: // localhost: 3000 nel tuo browser, vedrai una pagina di benvenuto di Rails, che mostra che la tua applicazione è ora in esecuzione.

Se genera un errore, potrebbero esserci diversi possibili problemi:

- C'è un problema con il `config/database.yml`
- Hai dipendenze nel tuo `Gemfile` che non sono state installate.
- Hai migrazioni in sospeso. Esegui `rails db:migrate`
- Nel caso in cui ci si sposta al `rails db:rollback` migrazione precedente `rails db:rollback`

Se ciò genera ancora un errore, allora dovresti controllare il tuo `config/database.yml`

## Crea una nuova app Rails con la tua scelta di database e includendo lo strumento di test RSpec

Rails utilizza `sqlite3` come database predefinito, ma è possibile generare una nuova applicazione rails con un database a scelta. Basta aggiungere l'opzione `-d` seguita dal nome del database.

```
$ rails new MyApp -T -d postgresql
```

Questa è una lista (non esaustiva) di opzioni di database disponibili:

- mysql
- oracolo
- PostgreSQL
- sqlite3
- FrontBase
- ibm_db
- server SQL
- jdbcmysql
- jdbcsqlite3
- jdbcpostgresql
- JDBC

Il comando -T indica di saltare l'installazione di minitest. Per installare una suite di test alternativa come RSpec , modifica il Gemfile e aggiungi

```
group :development, :test do
  gem 'rspec-rails',
end
```

Quindi avviare il seguente comando dalla console:

```
rails generate rspec:install
```

## Generazione di un controller

Per generare un controller (ad esempio `Posts` ), accedere alla directory del progetto da una riga di comando o terminale ed eseguire:

```
$ rails generate controller Posts
```

Puoi abbreviare questo codice sostituendo `generate` con `g` , ad esempio:

```
$ rails g controller Posts
```

Se apri l'app / controller / **posts_controller.rb** appena generata vedrai un controller senza azioni:

```
class PostsController < ApplicationController
    # empty
end
```

È possibile creare metodi predefiniti per il controller passando gli argomenti del nome del controller.

```
$ rails g controller ControllerName method1 method2
```

Per creare un controller all'interno di un modulo, specificare il nome del controller come percorso come `parent_module/controller_name` . Per esempio:

```
$ rails generate controller CreditCards open debit credit close
# OR
$ rails g controller CreditCards open debit credit close
```

Questo genererà i seguenti file:

```
Controller: app/controllers/credit_cards_controller.rb
Test:       test/controllers/credit_cards_controller_test.rb
Views:      app/views/credit_cards/debit.html.erb [...etc]
Helper:     app/helpers/credit_cards_helper.rb
```

Un controller è semplicemente una classe che è stata definita per ereditare da `ApplicationController` .

È all'interno di questa classe che definirai i metodi che diventeranno le azioni per questo

controller.

## Generare una risorsa con Scaffolds

Da guides.rubyonrails.org:

> Invece di generare direttamente un modello. . . sistemiamo un'impalcatura. Uno scaffold in Rails è un set completo di modello, migrazione del database per quel modello, controller per manipolarlo, visualizzazioni per visualizzare e manipolare i dati e una suite di test per ciascuno dei precedenti.

Ecco un esempio di scaffolding di una risorsa chiamata `Task` con un nome di stringa e una descrizione di testo:

```
rails generate scaffold Task name:string description:text
```

Questo genererà i seguenti file:

```
Controller: app/controllers/tasks_controller.rb
Test:       test/models/task_test.rb
            test/controllers/tasks_controller_test.rb
Routes:     resources :tasks added in routes.rb
Views:      app/views/tasks
            app/views/tasks/index.html.erb
            app/views/tasks/edit.html.erb
            app/views/tasks/show.html.erb
            app/views/tasks/new.html.erb
            app/views/tasks/_form.html.erb
Helper:     app/helpers/tasks_helper.rb
JS:         app/assets/javascripts/tasks.coffee
CSS:        app/assets/stylesheets/tasks.scss
            app/assets/stylesheets/scaffolds.scss
```

esempio per eliminare i file generati dallo scaffold per la risorsa chiamata `Task`

```
rails destroy scaffold Task
```

## Crea una nuova app di Rails con un adattatore per database non standard

Rails viene fornito di default con `ActiveRecord` , un ORM (Object Relational Mapping) derivato dal pattern con lo stesso nome .

Come un ORM, è costruito per gestire il mapping relazionale e, più precisamente, gestendo le richieste SQL per voi, quindi la limitazione solo ai database SQL.

Tuttavia, puoi ancora creare un'app Rails con un altro sistema di gestione del database:

1. crea semplicemente la tua app senza registrazione attiva

```
$ rails app new MyApp --skip-active-record
```

2. aggiungi il tuo sistema di gestione del database in `Gemfile`

```
gem 'mongoid', '~> 5.0'
```

3. `bundle install` e segui i passaggi di installazione dal database desiderato.

In questo esempio, `mongoid` è un object mapping per `MongoDB` e - come molte altre gemme di database create per rails - eredita anche da `ActiveModel` allo stesso modo di `ActiveRecord` , che fornisce un'interfaccia comune per molte funzionalità come convalide, callback, traduzioni, ecc. .

Altri adattatori di database includono, ma non sono limitati a:

- DataMapper
- sequel-rails

## Creazione di API Rails in JSON

Questo esempio presume che tu abbia esperienza nella creazione di applicazioni Rails.

Per creare un'app per sole API in Rails 5, esegui

```
rails new name-of-app --api
```

Aggiungi active_model_serializers in Gemfile

```
gem 'active_model_serializers'
```

installare il pacchetto nel terminale

```
bundle install
```

Impostare la scheda `ActiveModelSerializer` da utilizzare `:json_api`

```
# config/initializers/active_model_serializer.rb
ActiveModelSerializers.config.adapter = :json_api
Mime::Type.register "application/json", :json, %w( text/x-json application/jsonrequest
application/vnd.api+json )
```

Genera un nuovo scaffold per la tua risorsa

```
rails generate scaffold Task name:string description:text
```

Questo genererà i seguenti file:

Controller: app / controller / tasks_controller.rb

```
Test:       test/models/task_test.rb
            test/controllers/tasks_controller_test.rb
```

```
Routes:     resources :tasks added in routes.rb
Migration:  db/migrate/_create_tasks.rb
Model:      app/models/task.rb
Serializer: app/serializers/task_serializer.rb
Controller: app/controllers/tasks_controller.rb
```

## Installazione di Rails

### Installazione di Rails su Ubuntu

Su una ubuntu pulita, l'installazione di Rails dovrebbe essere diretta

Aggiornamento di pacchetti ubuntu

```
sudo apt-get update
sudo apt-get upgrade
```

Installa dipendenze Ruby e Rails

```
sudo apt-get install git-core curl zlib1g-dev build-essential libssl-dev libreadline-dev
libyaml-dev libsqlite3-dev sqlite3 libxml2-dev libxslt1-dev libcurl4-openssl-dev python-
software-properties libffi-dev
```

Installare il gestore delle versioni di ruby. In questo caso, quello facile sta usando rbenv

```
git clone https://github.com/rbenv/rbenv.git ~/.rbenv
echo 'export PATH="$HOME/.rbenv/bin:$PATH"' >> ~/.bashrc
echo 'eval "$(rbenv init -)"' >> ~/.bashrc
```

Installazione di Ruby Build

```
git clone https://github.com/rbenv/ruby-build.git ~/.rbenv/plugins/ruby-build
echo 'export PATH="$HOME/.rbenv/plugins/ruby-build/bin:$PATH"' >> ~/.bashrc
```

Riavvia Shell

```
exec $SHELL
```

Installa il rubino

```
rbenv install 2.3.1
rbenv global 2.3.1
rbenv rehash
```

Installazione delle guide

```
gem install rails
```

### Installazione di Rails su Windows

Passaggio 1: ***installazione di Ruby***

Abbiamo bisogno del linguaggio di programmazione Ruby installato. Possiamo usare una versione precompilata di Ruby chiamata RubyInstaller.

- Scarica ed esegui Ruby Installer da rubyinstaller.org .
- Esegui il programma di installazione. Seleziona "Aggiungi eseguibili Ruby al tuo PATH", quindi installa.
- Per accedere a Ruby, vai al menu di Windows, fai clic su Tutti i programmi, scorri verso il basso fino a Ruby e fai clic su "Start Command Prompt with Ruby". Si aprirà un terminale del prompt dei comandi. Se digiti `ruby -v` e premi Invio, dovresti vedere il numero di versione di Ruby che hai installato.

Passaggio 2: ***Kit di sviluppo di rubino***

Dopo aver installato Ruby, possiamo provare a installare Rails. Ma alcune delle librerie di Rails dipendono dalla necessità di alcuni strumenti di compilazione per poter essere compilate e Windows manca di tali strumenti per impostazione predefinita. È possibile identificarlo se si riscontra un errore durante il tentativo di installare Rails `Gem::InstallError: The '[gem name]' native gem requires installed build tools.` Per risolvere questo problema, è necessario installare il Kit di sviluppo di Ruby.

- Scarica il DevKit
- Esegui il programma di installazione.
- Dobbiamo specificare una cartella in cui installeremo il DevKit in modo permanente. Consiglio di installarlo nella root del tuo disco rigido, in `C:\RubyDevKit` . (Non utilizzare spazi nel nome della directory.)

Ora dobbiamo rendere disponibili gli strumenti DevKit a Ruby.

- Nel prompt dei comandi, passare alla directory DevKit. `cd C:\RubyDevKit` o qualsiasi altra directory in cui l'hai installato.
- Abbiamo bisogno di eseguire uno script Ruby per inizializzare l'installazione di DevKit. Digitare `ruby dk.rb init` . Ora diremo quello stesso script per aggiungere il DevKit alla nostra installazione di Ruby. Digitare `ruby dk.rb install` .

DevKit ora dovrebbe essere disponibile per gli strumenti di Ruby da utilizzare quando si installano nuove librerie.

Passaggio 3: ***Rails***

Ora possiamo installare Rails. Rails viene fornito come una gemma di Ruby. Nel tuo prompt dei comandi, digita:

```
gem install rails
```

Una volta premuto Invio, il programma `gem` scaricherà e installerà quella versione della gemma Rails, insieme a tutte le altre gemme di cui Rails dipende.

Passaggio 4: ***Node.js***

Alcune librerie da cui dipende Rails richiedono l'installazione di un runtime JavaScript. Installiamo Node.js in modo che quelle librerie funzionino correttamente.

- Scarica il programma di installazione di Node.js da **qui** .
- Al termine del download, visitare la cartella dei download ed eseguire il `node-v4.4.7.pkg` installazione `node-v4.4.7.pkg` .
- Leggere il contratto di licenza completo, accettare i termini e fare clic su Avanti attraverso il resto della procedura guidata, lasciando tutto come predefinito.
- Potrebbe apparire una finestra che chiede se si desidera consentire all'applicazione di apportare modifiche al computer. Fai clic su "Sì".
- Al termine dell'installazione, sarà necessario riavviare il computer in modo che Rails possa accedere a Node.js.

Una volta riavviato il computer, non dimenticare di andare al menu di Windows, fare clic su "Tutti i programmi", scorrere verso il basso fino a Ruby e fare clic su "Start Command Prompt with Ruby".

Leggi Iniziare con Ruby on Rails online: https://riptutorial.com/it/ruby-on-rails/topic/225/iniziare-con-ruby-on-rails

# Capitolo 2: ActionCable

## Osservazioni

ActionCable era disponibile per Rails 4.x, ed era in bundle in Rails 5. Permette un facile utilizzo dei websocket per la comunicazione in tempo reale tra server e client.

## Examples

### [Base] Lato server

```
# app/channels/appearance_channel.rb
class NotificationsChannel < ApplicationCable::Channel
  def subscribed
    stream_from "notifications"
  end

  def unsubscribed
  end

  def notify(data)
    ActionCable.server.broadcast "notifications", { title: 'New things!', body: data }
  end
end
```

### [Base] Lato client (Coffeescript)

# app / beni / javascript / canali / notifications.coffee

```
App.notifications = App.cable.subscriptions.create "NotificationsChannel",
  connected: ->
    # Called when the subscription is ready for use on the server
    $(document).on "change", "input", (e)=>
      @notify(e.target.value)

  disconnected: ->
    # Called when the subscription has been terminated by the server
     $(document).off "change", "input"

  received: (data) ->
    # Called when there's incoming data on the websocket for this channel
    $('body').append(data)

  notify: (data)->
    @perform('notify', data: data)
```

# app / assets / javascripts / application.js # di solito generato in questo modo

```
//= require jquery
//= require jquery_ujs
//= require turbolinks
//= require_tree .
```

# app / assets / javascripts / cable.js # di solito generato in questo modo

```
//= require action_cable
//= require_self
//= require_tree ./channels

(function() {
  this.App || (this.App = {});

  App.cable = ActionCable.createConsumer();

}).call(this);
```

### Autenticazione utente

```
# app/channels/application_cable/connection.rb
module ApplicationCable
  class Connection < ActionCable::Connection::Base
    identified_by :current_user

    def connect
      self.current_user = find_verified_user
      logger.add_tags 'ActionCable', current_user.id
      # Can replace current_user.id with usernames, ids, emails etc.
    end

    protected

    def find_verified_user
      if verified_user = env['warden'].user
        verified_user
      else
        reject_unauthorized_connection
      end
    end
  end
end
```

Leggi ActionCable online: https://riptutorial.com/it/ruby-on-rails/topic/1498/actioncable

# Capitolo 3: ActionController

## introduzione

Action Controller è la C in MVC. Dopo che il router ha determinato quale controller utilizzare per una richiesta, il controller è responsabile di dare un senso alla richiesta e produrre l'output.

Il controller riceverà la richiesta, recupera o salva i dati da un modello e utilizza una vista per creare output. Un controller può essere pensato come un intermediario tra modelli e viste. Rende i dati del modello disponibili alla vista in modo che possano essere visualizzati dall'utente e salva o aggiorna i dati dell'utente sul modello.

## Examples

### Output JSON anziché HTML

```
class UsersController < ApplicationController
  def index
    hashmap_or_array = [{ name: "foo", email: "foo@example.org" }]

    respond_to do |format|
      format.html { render html: "Hello World" }
      format.json { render json: hashmap_or_array }
    end
  end
end
```

Inoltre avrai bisogno del percorso:

```
resources :users, only: [:index]
```

Ciò risponderà in due modi diversi alle richieste su `/users` :

- Se visiti `/users` o `/users.html` , mostrerà una pagina html con il contenuto `Hello World`
- Se visiti `/users.json` , verrà visualizzato un oggetto JSON contenente:

```
[
  {
    "name": "foo",
    "email": "foo@example.org"
  }
]
```

Puoi **omettere** `format.html { render inline: "Hello World" }` se vuoi assicurarti che il tuo percorso risponda solo alle richieste JSON.

### Controller (base)

```
class UsersController < ApplicationController
  def index
    respond_to do |format|
      format.html { render html: "Hello World" }
    end
  end
end
```

Questo è un controller di base, con l'aggiunta del seguente percorso (in routes.rb):

```
resources :users, only: [:index]
```

Visualizzerà il messaggio `Hello World` in una pagina Web quando si accede all'URL `/users`

## parametri

I controllori hanno accesso ai parametri HTTP (potresti conoscerli come `?name=foo` negli URL, ma Ruby on Rails gestisce anche diversi formati!) E genera risposte diverse basate su di essi. Non c'è un modo per distinguere tra i parametri GET e POST, ma in ogni caso non dovresti farlo.

```
class UsersController < ApplicationController
  def index
    respond_to do |format|
      format.html do
        if params[:name] == "john"
          render html: "Hello John"
        else
          render html: "Hello someone"
        end
      end
    end
  end
end
```

Come al solito il nostro percorso:

```
resources :users, only: [:index]
```

Accedi all'URL `/users?name=john` e l'output sarà `Hello John` , access `/users?name=whatever` e l'output sarà `Hello someone`

## Parametri di filtraggio (base)

```
class UsersController < ApplicationController
  def index
    respond_to do |format|
      format.html do
        render html: "Hello #{ user_params[:name] } user_params[:sentence]"
      end
    end
  end

  private
```

```
  def user_params
    if params[:name] == "john"
      params.permit(:name, :sentence)
    else
      params.permit(:name)
    end
  end
end
```

Puoi consentire (o rifiutare) alcuni parametri in modo che solo quello che vuoi *passi* e che non abbia brutte sorprese come le opzioni di impostazione dell'utente che non devono essere modificate.

Visiting `/users?name=john&sentence=developer` visualizzerà `Hello john developer` , tuttavia visitando `/users?name=smith&sentence=spy` mostrerà solo `Hello smith` , perché `:sentence` è consentita solo quando accedi come `john`

## Riorientare

Supponendo il percorso:

```
resources :users, only: [:index]
```

Puoi reindirizzare a un URL diverso usando:

```
class UsersController
  def index
    redirect_to "http://stackoverflow.com/"
  end
end
```

Puoi tornare alla pagina precedente che l'utente ha visitato utilizzando:

```
redirect_to :back
```

Nota che in *Rails 5* la sintassi per il reindirizzamento è diversa:

```
redirect_back fallback_location: "http://stackoverflow.com/"
```

Che proverà a reindirizzare alla pagina precedente e nel caso non sia possibile (il browser sta bloccando l'intestazione HTTP_REFERRER), verrà reindirizzato a `:fallback_location`

## Utilizzo di viste

Supponendo il percorso:

```
resources :users, only: [:index]
```

E il controller:

```
class UsersController < ApplicationController
  def index
    respond_to do |format|
      format.html { render }
    end
  end
end
```

L' `app/users/index.html.erb` visualizzazione `app/users/index.html.erb` sarà renderizzata. Se la vista è:

```
Hello <strong>World</strong>
```

L'output sarà una pagina web con il testo: "Hello **World** "

Se vuoi rendere una vista diversa, puoi usare:

```
render "pages/home"
```

Verranno invece utilizzati i file `app/views/pages/home.html.erb` .

È possibile passare variabili alle viste utilizzando le variabili di istanza del controllore:

```
class UsersController < ApplicationController
  def index
    @name = "john"

    respond_to do |format|
      format.html { render }
    end
  end
end
```

E nel file `app/views/users/index.html.erb` puoi usare `@name` :

```
Hello <strong><%= @name %></strong>
```

E l'output sarà: "Ciao **John** "

Una nota importante intorno alla sintassi del rendering, è che puoi omettere completamente la sintassi del `render` , Rails presume che se lo ometti. Così:

```
class UsersController < ApplicationController
  def index
    respond_to do |format|
      format.html { render }
    end
  end
end
```

Può essere scritto invece come:

```
class UsersController < ApplicationController
  def index
    respond_to do |format|
      format.html
    end
  end
end
```

Rails è abbastanza intelligente da capire che deve rendere il file `app/views/users/index.html.erb` .

## 404 quando la registrazione non è stata trovata

Salvataggio da un errore non trovato di record invece di mostrare un'eccezione o una pagina bianca:

```
class ApplicationController < ActionController::Base

  # ... your other stuff here

  rescue_from ActiveRecord::RecordNotFound do |exception|
    redirect_to root_path, 404, alert: 'Record not found'
  end
end
```

## Controller REST di base

```
class PostsController < ApplicationController
  before_action :set_post, only: [:show, :edit, :update, :destroy]

  def index
    @posts = Post.all
  end

  def show

  end

  def new
    @post = Post.new
  end

  def edit

  end

  def create
    @post = Post.new(post_params)

    respond_to do |format|
      if @post.save
        format.html { redirect_to @post, notice: 'Post was successfully created.' }
        format.json { render :show, status: :created, location: @post }
      else
        format.html { render :new }
        format.json { render json: @post.errors, status: :unprocessable_entity }
      end
```

```
    end
  end

  def update
    respond_to do |format|
      if @post.update(post_params)
        format.html { redirect_to @post.company, notice: 'Post was successfully updated.' }
        format.json { render :show, status: :ok, location: @post }
      else
        format.html { render :edit }
        format.json { render json: @post.errors, status: :unprocessable_entity }
      end
    end
  end

  def destroy
    @post.destroy
    respond_to do |format|
      format.html { redirect_to posts_url, notice: 'Post was successfully destroyed.' }
      format.json { head :no_content }
    end
  end

  private
    def set_post
      @post = Post.find(params[:id])
    end

    def post_params
      params.require(:post).permit(:title, :body, :author)
    end
end
```

## Visualizza le pagine di errore per le eccezioni

Se vuoi mostrare agli utenti errori significativi invece di "scusa, qualcosa è andato storto", Rails ha una buona utilità per lo scopo.

Apri l' `app/controllers/application_controller.rb` file `app/controllers/application_controller.rb` e dovresti trovare qualcosa di simile a questo:

```
class ApplicationController < ActionController::Base
  protect_from_forgery with: :exception
end
```

Ora possiamo aggiungere un `rescue_from` per recuperare da errori specifici:

```
class ApplicationController < ActionController::Base
  protect_from_forgery with: :exception
  rescue_from ActiveRecord::RecordNotFound, with: :record_not_found

  private

  def record_not_found
    render html: "Record <strong>not found</strong>", status: 404
  end
end
```

Si raccomanda di non eseguire il salvataggio da `Exception` o `StandardError` altrimenti Rails non sarà in grado di visualizzare pagine utili in caso di errori.

## filtri

I filtri sono metodi che vengono eseguiti "prima", "dopo" o "attorno" a un'azione del controller. Sono ereditati, quindi se ne inserisci uno nel tuo `ApplicationController` verranno eseguiti per ogni richiesta ricevuta dalla tua applicazione.

**Prima del filtro**

Prima che i filtri vengano eseguiti prima dell'azione del controllore e possono interrompere la richiesta (e / o il reindirizzamento). Un uso comune è verificare se un utente è loggato:

```
class ApplicationController < ActionController::Base
  before_action :authenticate_user!

  def authenticate_user!
    redirect_to some_path unless user_signed_in?
  end
end
```

Prima che i filtri vengano eseguiti sulle richieste prima che la richiesta raggiunga l'azione del controllore. Può restituire una risposta e ignorare completamente l'azione.

Altri usi comuni dei filtri precedenti convalidano l'autenticazione dell'utente prima di concedere loro l'accesso all'azione designata per gestire la loro richiesta. Li ho visti anche per caricare una risorsa dal database, controllare le autorizzazioni su una risorsa o gestire i reindirizzamenti in altre circostanze.

**Dopo il filtro**

Dopo che i filtri sono simili a quelli "precedenti", ma quando vengono eseguiti dopo l'esecuzione dell'azione, hanno accesso all'oggetto risposta che sta per essere inviato. Quindi, in breve, i filtri vengono eseguiti al termine dell'azione. Può modificare la risposta. La maggior parte delle volte se qualcosa viene fatto in un after filter, può essere fatto nell'azione stessa, ma se c'è qualche logica da eseguire dopo aver eseguito una serie di azioni, allora un after filter è un buon posto dove fare esso.

In generale, ho visto dopo e intorno ai filtri utilizzati per la registrazione.

**Intorno al filtro**

Intorno ai filtri potrebbe essere presente logica prima e dopo l'esecuzione dell'azione. Si arrende semplicemente all'azione in qualunque posto sia necessario. Si noti che non ha bisogno di arrendersi all'azione e può essere eseguito senza farlo come un filtro precedente.

Intorno ai filtri è responsabile l'esecuzione delle azioni associate cedendo, analogamente a come funzionano i media di Rack.

Intorno ai callback avvolgono l'esecuzione delle azioni. Puoi scrivere un callback in due stili diversi. Nel primo, il callback è un singolo pezzo di codice. Questo codice viene chiamato prima che l'azione venga eseguita. Se il codice di richiamata richiama rendimento, l'azione viene eseguita. Al termine dell'azione, il codice di richiamata continua ad essere eseguito. Pertanto, il codice prima del rendimento è come un callback prima dell'azione e il codice dopo il rendimento è il callback dopo l'azione. Se il codice di richiamata non invoca mai il rendimento. l'azione non viene eseguita, equivale ad avere una prima azione callback return false.

Ecco un esempio del filtro in giro:

```
around_filter :catch_exceptions

private
  def catch_exceptions
    begin
      yield
    rescue Exception => e
      logger.debug "Caught exception! #{e.message}"
    end
  end
```

Questo prenderà l'eccezione di qualsiasi azione e metterà il messaggio nel tuo log. È possibile utilizzare i filtri per la gestione delle eccezioni, l'installazione e la rimozione e una miriade di altri casi.

### Solo ed Escluso

Tutti i filtri possono essere applicati a azioni specifiche, usando `:only` e `:except` :

```
class ProductsController < ApplicationController
  before_action :set_product, only: [:show, :edit, :update]

  # ... controller actions

  # Define your filters as controller private methods
  private

  def set_product
    @product = Product.find(params[:id])
  end
end
```

### Saltare il filtro

Tutti i filtri (anche quelli ereditati) possono essere saltati per alcune azioni specifiche:

```
class ApplicationController < ActionController::Base
  before_action :authenticate_user!

  def authenticate_user!
    redirect_to some_path unless user_signed_in?
  end
end
```

```
class HomeController < ApplicationController
  skip_before_action :authenticate_user!, only: [:index]

  def index
  end
end
```

Poiché vengono ereditati, i filtri possono anche essere definiti in un controller "parent" di un `namespace` . Supponiamo, ad esempio, di avere uno spazio dei nomi di `admin` e, naturalmente, si desidera che solo gli utenti amministratori possano accedervi. Potresti fare qualcosa del genere:

```
# config/routes.rb
namespace :admin do
  resources :products
end

# app/controllers/admin_controller.rb
class AdminController < ApplicationController
  before_action :authenticate_admin_user!

  private

  def authenticate_admin_user!
    redirect_to root_path unless current_user.admin?
  end
end

# app/controllers/admin/products_controller.rb
class Admin::ProductsController < AdminController
  # This controller will inherit :authenticate_admin_user! filter
end
```

`before_filter` attenzione che in **Rails 4.x** puoi usare `before_filter` insieme a `before_action` , ma `before_filter` è attualmente deprecato in **Rails 5.0.0** e verrà rimosso in **5.1** .

## Generazione di un controller

Rails fornisce molti generatori, ovviamente anche per i controller.

È possibile generare un nuovo controller eseguendo questo comando nella cartella dell'app

```
rails generate controller NAME [action action] [options]
```

*Nota: è anche possibile utilizzare l'alias `rails g` per invocare la `rails generate`*

Ad esempio, per generare un controller per un modello di `Product` , con le azioni `#index` e `#show` eseguite

```
rails generate controller products index show
```

Questo creerà il controller in `app/controllers/products_controller.rb` , con entrambe le azioni specificate

```
class ProductsController < ApplicationController
  def index
  end

  def show
  end
end
```

Sarà inoltre possibile creare un `products` cartella all'interno `app/views/` , che contiene i due modelli per le azioni del controller (cioè `index.html.erb` e `show.html.erb` , *notare che l'estensione può variare a seconda del motore di template, quindi se si Usando `slim` , ad esempio, il generatore creerà `index.html.slim` e `show.html.slim`* )

Inoltre, se hai specificato azioni, verranno aggiunte al tuo file di `routes`

```
# config/routes.rb
get 'products/show'
get 'products/index'
```

Rails crea un file di supporto per te, in `app/helpers/products_helper.rb` , e anche i file delle risorse in `app/assets/javascripts/products.js` e `app/assets/stylesheets/products.css` . Per quanto riguarda le visualizzazioni, il generatore modifica questo comportamento in base a quanto specificato nel tuo `Gemfile` : cioè, se stai utilizzando `Coffeescript` e `Sass` nella tua applicazione, il generatore di controller genererà invece `products.coffee` e `products.sass` .

Infine, ma non meno importante, Rails genera anche file di test per il controller, l'helper e le visualizzazioni.

Se non vuoi che qualcuno di questi venga creato per te, puoi dire a Rails di saltarli, basta anteporre qualsiasi opzione a

`--no-` o `--skip` , come questo:

```
rails generate controller products index show --no-assets --no-helper
```

E il generatore salterà sia le `assets` che l' `helper`

Se devi creare un controller per uno **`namespace`** dei **`namespace`** specifico, aggiungilo davanti a `NAME` :

```
rails generate controller admin/products
```

Questo creerà il controller all'interno di `app/controllers/admin/products_controller.rb`

Rails può anche generare un controller RESTful completo per te:

```
rails generate scaffold_controller MODEL_NAME # available from Rails 4
rails generate scaffold_controller Product
```

## Salvataggio di ActiveRecord :: RecordNotFound con redirect_to

Puoi salvare un'eccezione RecordNotFound con un reindirizzamento invece di mostrare una pagina di errore:

```
class ApplicationController < ActionController::Base

  # your other stuff

  rescue_from ActiveRecord::RecordNotFound do |exception|
    redirect_to root_path, 404, alert: I18n.t("errors.record_not_found")
  end
end
```

Leggi ActionController online: https://riptutorial.com/it/ruby-on-rails/topic/2838/actioncontroller

# Capitolo 4: ActionMailer

## introduzione

Action Mailer ti consente di inviare e-mail dalla tua applicazione utilizzando le classi e le viste del mailer. I mailer funzionano in modo molto simile ai controller. Essi ereditano da ActionMailer :: Base e vivono in app / mailer e hanno viste associate che appaiono in app / view.

## Osservazioni

Si consiglia di elaborare l'invio di e-mail in modo asincrono in modo da non legare il server web. Questo può essere fatto attraverso vari servizi come `delayed_job` .

## Examples

### Mailer di base

Questo esempio utilizza quattro file diversi:

- Il modello utente
- Il mailer dell'utente
- Il modello html per l'email
- Il modello di testo semplice per l'email

In questo caso, il modello utente chiama il metodo `approved` nel mailer e passa il `post` che è stato approvato (il metodo `approved` nel modello può essere richiamato da un callback, da un metodo controller, ecc.). Quindi, il mailer genera l'email dal modello html o dal testo normale utilizzando le informazioni del `post` passato (ad esempio il titolo). Per impostazione predefinita, il mailer utilizza il modello con lo stesso nome del metodo nel mailer (motivo per cui sia il metodo mailer che i modelli hanno il nome "approvato").

# user_mailer.rb

```
class UserMailer < ActionMailer::Base
  default from: "donotreply@example.com"

  def approved(post)
      @title = post.title
      @user = post.user
      mail(to: @user.email, subject: "Your Post was Approved!")
  end
end
```

# user.rb

```
def approved(post)
    UserMailer.approved(post)
end
```

# approved.html.erb

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
    <head>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
      <title>Post Approved</title>
    </head>
    <body>
        <h2>Congrats <%= @user.name %>! Your post (#<%= @title %>) has been approved!</h2>
        <p>We look forward to your future posts!</p>
    </body>
</html>
```

# approved.text.erb

```
Congrats <%= @user.name %>! Your post (#<%= @title %>) has been approved!
We look forward to your future posts!
```

## Generare un nuovo mailer

Per generare un nuovo mailer, immettere il seguente comando

```
rails generate mailer PostMailer
```

Questo genererà un file modello vuoto in `app/mailers/post_mailer.rb` chiamato *PostMailer*

```
class PostMailer < ApplicationMailer
end
```

Verranno inoltre generati due file di layout per la vista e-mail, uno per il formato html e uno per il formato testo.

Se si preferisce non utilizzare il generatore, è possibile creare i propri mailer. Assicurati che ereditino da `ActionMailer::Base`

## Aggiunta di allegati

`ActionMailer` consente anche di allegare file.

```
attachments['filename.jpg'] = File.read('/path/to/filename.jpg')
```

Per impostazione predefinita, gli allegati saranno codificati con `Base64` . Per cambiare questo, puoi aggiungere un hash al metodo degli allegati.

```
attachments['filename.jpg'] = {
  mime_type: 'application/gzip',
  encoding: 'SpecialEncoding',
  content: encoded_content
}
```

Puoi anche aggiungere allegati in linea

```
attachments.inline['image.jpg'] = File.read('/path/to/image.jpg')
```

## Callback di ActionMailer

ActionMailer supporta tre callback

- before_action
- after_action
- around_action

Forniscili nella tua classe Mailer

```
class UserMailer < ApplicationMailer
  after_action :set_delivery_options, :prevent_delivery_to_guests, :set_business_headers
```

Quindi creare questi metodi sotto la parola chiave `private`

```
private
  def set_delivery_options
  end

  def prevent_delivery_to_guests
  end

  def set_business_headers
  end
end
```

## Generare una Newsletter pianificata

Crea il modello di **newsletter** :

```
  rails g model Newsletter name:string email:string

  subl app/models/newsletter.rb

  validates :name, presence: true
  validates :email, presence: true
```

Crea il controller della **newsletter** :

```
rails g controller Newsletters create

class NewslettersController < ApplicationController
  skip_before_action :authenticate_user!
  before_action :set_newsletter, only: [:destroy]

  def create
    @newsletter = Newsletter.create(newsletter_params)
    if @newsletter.save
      redirect_to blog_index_path
    else
      redirect_to root_path
    end
  end

  private

    def set_newsletter
      @newsletter = Newsletter.find(params[:id])
    end

    def newsletter_params
      params.require(:newsletter).permit(:name, :email)
    end

end
```

Dopodiché, modifica la vista **create.html.erb** sul nome nex. Convertiremo questo file e la **vista parziale** che verrà memorizzata all'interno del **piè di pagina** . Il nome sarà **_form.html.erb** .

| Cambia il nome del file da: | A: |
| --- | --- |
| app / views / **newsletters / create.html.erb** | app / views / **newsletters / _form.html.erb** |

Successivamente, imposta i percorsi:

```
subl app/config/routes.rb

resources :newsletters
```

In seguito, dobbiamo impostare il modulo che useremo per salvare ogni mail:

```
subl app/views/newsletters/_form.html.erb

<%= form_for (Newsletter.new) do |f| %>
  <div class="col-md-12" style="margin: 0 auto; padding: 0;">
    <div class="col-md-6" style="padding: 0;">
      <%= f.text_field :name, class: 'form-control', placeholder:'Nombre' %>
    </div>
    <div class="col-md-6" style="padding: 0;">
      <%= f.text_field :email, class: 'form-control', placeholder:'Email' %>
    </div>
  </div>
  <div class="col-md-12" style="margin: 0 auto; padding:0;">
```

```
      <%= f.submit class:"col-md-12 tran3s s-color-bg hvr-shutter-out-horizontal",
style:'border: none; color: white; cursor: pointer; margin: 0.5em auto; padding: 0.75em;
width: 100%;' %>
    </div>
  <% end %>
```

E dopo, inserisci sul footer:

```
  subl app/views/layouts/_footer.html.erb

  <%= render 'newsletters/form' %>
```

Ora, installa - **letter_opener** - per poter visualizzare l'anteprima dell'email nel browser predefinito invece di inviarlo. Ciò significa che non è necessario impostare la consegna delle e-mail nel proprio ambiente di sviluppo e non è più necessario preoccuparsi di inviare accidentalmente una e-mail di prova all'indirizzo di qualcun altro.

Per prima cosa aggiungi la gemma al tuo ambiente di sviluppo ed esegui il comando bundle per installarlo.

```
  subl your_project/Gemfile

  gem "letter_opener", :group => :development
```

Quindi imposta il metodo di consegna nell'ambiente di sviluppo:

```
  subl your_project/app/config/environments/development.rb

  config.action_mailer.delivery_method = :letter_opener
```

Ora, crea una **struttura Mailer** per gestire gli interi mailer che lavoreremo. Nel terminale

```
  rails generate mailer UserMailer newsletter_mailer
```

E all'interno **dell'UtenteMailer** , dobbiamo creare un metodo chiamato **Newsletter Mailer** che verrà creato per contenere all'interno l'ultimo post del blog e verrà attivato con un'azione di rake. Assumeremo che tu avessi una struttura del blog creata prima.

```
subl your_project/app/mailers/user_mailer.rb

class UserMailer 'your_gmail_account@gmail.com'


  def newsletter_mailer
    @newsletter = Newsletter.all
    @post = Post.last(3)
    emails = @newsletter.collect(&:email).join(", ")
    mail(to: emails, subject: "Hi, this is a test mail.")
  end

end
```

Successivamente, crea il **modello di posta elettronica** :

```
subl your_project/app/views/user_mailer/newsletter_mailer.html.erb

<p> Dear Followers: </p>
<p> Those are the lastest entries to our blog. We invite you to read and share everything we
did on this week. </p>

<br/>
<table>
<% @post.each do |post| %>
  <%#= link_to blog_url(post) do %>
      <tr style="display:flex; float:left; clear:both;">
        <td style="display:flex; float:left; clear:both; height: 80px; width: 100px;">
          <% if post.cover_image.present? %>
            <%= image_tag post.cover_image.fullsize.url, class:"principal-home-image-slider"
%>
          <%# else %>
            <%#= image_tag 'http://your_site_project.com' + post.cover_video,
class:"principal-home-image-slider" %>
            <%#= raw(video_embed(post.cover_video)) %>
          <% end %>
        </td>
        <td>
          <h3>
            <%= link_to post.title, :controller => "blog", :action => "show", :only_path =>
false, :id => post.id %>
          </h3>
          <p><%= post.subtitle %></p>
        </td>
        <td style="display:flex; float:left; clear:both;">

        </td>
      </tr>
  <%# end %>
<% end %>
</table>
```

Poiché desideriamo inviare l'email come processo separato, creiamo un'attività Rake per attivare l'email. Aggiungi un nuovo file chiamato email_tasks.rake alla directory lib / tasks dell'applicazione Rails:

```
touch lib/taks/email_tasks.rake

desc 'weekly newsletter email'
task weekly_newsletter_email: :environment do
  UserMailer.newsletter_mailer.deliver!
end
```

Send_digest_email:: environment significa caricare l'ambiente Rails prima di eseguire l'attività, in modo da poter accedere alle classi dell'applicazione (come UserMailer) all'interno dell'attività.

Ora, eseguendo il comando rake -T verrà elencata l'attività Rake appena creata. Prova tutto funziona eseguendo l'attività e controllando se l'e-mail è stata inviata o meno.

Per verificare se il metodo mailer funziona, eseguire il comando rake:

```
  rake weekly_newsletter_email
```

A questo punto, abbiamo un'attività di rake di lavoro che può essere pianificata usando **crontab** . Quindi installeremo il **Whenever Gem** che viene utilizzato per fornire una sintassi chiara per la scrittura e la distribuzione di cron jobs.

```
subl your_project/Gemfile

  gem 'whenever', require: false
```

Successivamente, esegui il comando successivo per creare un file config / schedule.rb iniziale per te (purché la cartella di configurazione sia già presente nel tuo progetto).

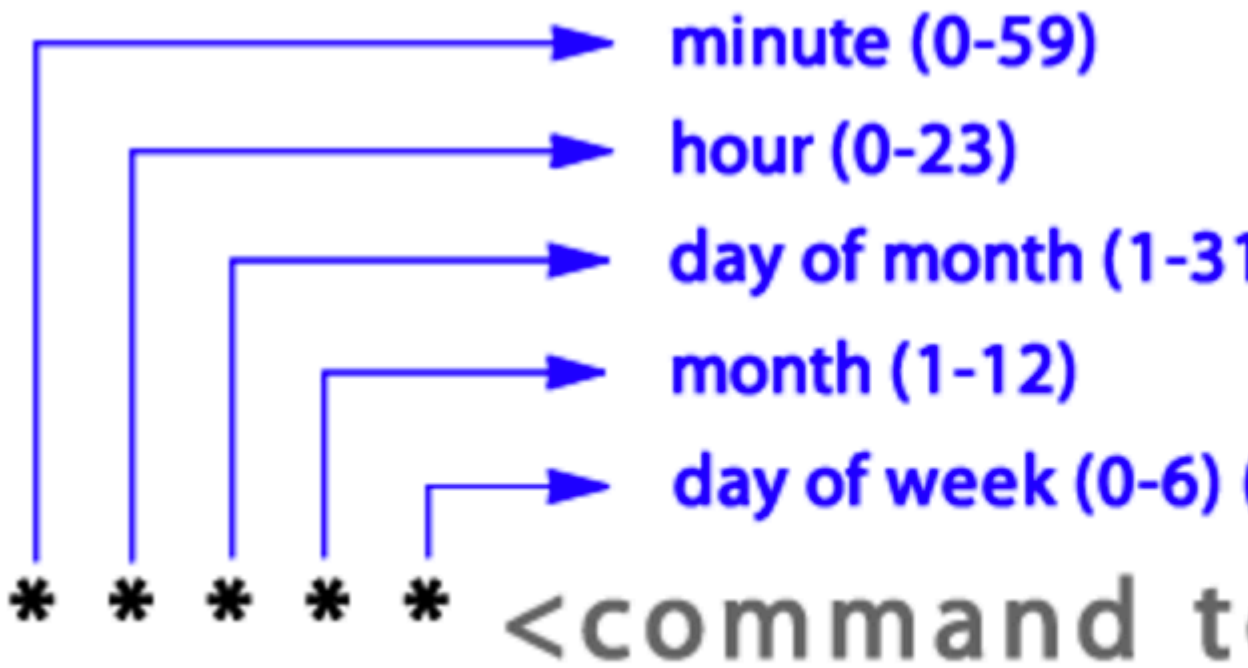
```
  wheneverize .

  [add] writing `./config/schedule.rb'
  [done] wheneverized!
```

Ora, all'interno del file di pianificazione, dobbiamo creare il nostro **CRON JOB** e chiamare il metodo mailer nel determinare il LAVORO CRON per eseguire alcune attività senza assistenza e in un intervallo di tempo selezionato. È possibile utilizzare diversi tipi di sintassi come spiegato in questo collegamento .

```
subl your_project/config/schedule.rb

every 1.day, :at => '4:30 am' do
  rake 'weekly_newsletter_email'
end
```


minute (0-59)
hour (0-23)
day of month (1-31
month (1-12)
day of week (0-6)
* * * * *
<command t

```
every 3.hours do # 1.minute 1.day 1.week 1.m
  runner "MyModel.some_process"
  rake "my:rake:task"
  command "/usr/bin/my_great_command"
end

every 1.day, :at => '4:30 am' do
  runner "MyModel.task_to_run_at_four_thirty
end

every :hour do # Many shortcuts available: :
  runner "SomeModel.ladeeda"
end

every :sunday, :at => '12pm' do # Use any da
  runner "Task.do_something_great"
end

every '0 0 27-31 * *' do
  command "echo 'you can use raw cron syntax
end

# run this task only on servers with the :ap
# see Capistrano roles section below
every :day, :at => '12:20am', :roles => [:ap
  rake "app_server:task"
end
```

stato creato con successo possiamo usare il prossimo comando da leggere dal terminale, il nostro lavoro programmato in CRON SYNTAX:

```
your_project your_mac_user$ whenever

30 4 * * * /bin/bash -l -c 'cd /Users/your_mac_user/Desktop/your_project &&
RAILS_ENV=production bundle exec rake weekly_newsletter_email --silent'
```

Ora, per eseguire il test in Development Environment, è consigliabile impostare la riga successiva sul file principal **application.rb** per consentire all'applicazione di sapere dove si trovano i modelli che utilizzerà.

```
  subl your_project/config/application.rb

  config.action_mailer.default_url_options = { :host => "http://localhost:3000/" }
```

Ora per consentire a **Capistrano V3 di** salvare il nuovo **Cron Job** all'interno del server e il trigger che ha attivato l'esecuzione di questa attività, dobbiamo aggiungere il prossimo requisito:

```
  subl your_project/Capfile

  require 'whenever/capistrano'
```

E inserire nel file di **distribuzione** l'identificativo che **CRON JOB** utilizzerà per l' **ambiente** e il nome **dell'applicazione** .

```
subl your_project/config/deploy.rb

set :whenever_identifier, ->{ "#{fetch(:application)}_#{fetch(:rails_env)}" }
```

E pronto, dopo aver salvato le modifiche su ogni file, esegui il comando capistrano deploy:

```
  cap production deploy
```

E ora il tuo JOB è stato creato e calendarizzato per eseguire il Metodo Mailer che è quello che voglio e nel tempo che abbiamo impostato su questi file.

## ActionMailer Interceptor

Action Mailer fornisce hook nei metodi dell'intercettore. Questi ti consentono di registrare le classi che vengono chiamate durante il ciclo di vita della consegna della posta.

Una classe interceptor deve implementare il metodo: delivering_email (message) che verrà chiamato prima dell'invio dell'e-mail, consentendo di apportare modifiche all'e-mail prima che colpisca gli agenti di consegna. La classe dovrebbe apportare eventuali modifiche necessarie direttamente all'istanza di Mail :: Message passata.

Può essere utile per gli sviluppatori inviare e-mail a se stessi non utenti reali.

Esempio di registrazione di un interceptor di Actionmailer:

```
# config/initializers/override_mail_recipient.rb

if Rails.env.development? or Rails.env.test?
  class OverrideMailRecipient
    def self.delivering_email(mail)
      mail.subject = 'This is dummy subject'
      mail.bcc = 'test_bcc@noemail.com'
      mail.to = 'test@noemail.com'
    end
  end
  ActionMailer::Base.register_interceptor(OverrideMailRecipient)
end
```

Leggi ActionMailer online: https://riptutorial.com/it/ruby-on-rails/topic/2481/actionmailer

# Capitolo 5: ActiveJob

## introduzione

Active Job è un framework per la dichiarazione dei lavori e la loro esecuzione su una varietà di backend di accodamento. Questi lavori possono essere qualsiasi cosa, da pulizie pianificate regolarmente, a spese di fatturazione, a spedizioni. Tutto ciò che può essere tagliato in piccole unità di lavoro e correre in parallelo, davvero.

## Examples

### Crea il lavoro

```
class GuestsCleanupJob < ApplicationJob
  queue_as :default

  def perform(*guests)
    # Do something later
  end
end
```

### Accendi il lavoro

```
# Enqueue a job to be performed as soon as the queuing system is free.
GuestsCleanupJob.perform_later guest
```

Leggi ActiveJob online: https://riptutorial.com/it/ruby-on-rails/topic/8996/activejob

# Capitolo 6: ActiveModel

## Osservazioni

ActiveModel è stato creato per estrarre il comportamento del modello di ActiveRecord in una preoccupazione separata. Questo ci consente di utilizzare il comportamento di ActiveModel in qualsiasi oggetto, non solo nei modelli ActiveRecord.

Gli oggetti ActiveRecord includono tutto questo comportamento per impostazione predefinita.

## Examples

### Utilizzo di ActiveModel :: Validations

Puoi convalidare qualsiasi oggetto, anche il semplice rubino.

```
class User
  include ActiveModel::Validations

  attr_reader :name, :age

  def initialize(name, age)
    @name = name
    @age  = age
  end

  validates :name, presence: true
  validates :age, numericality: { only_integer: true, greater_than: 12 }
end
```

```
User.new('John Smith', 28).valid? #=> true
User.new('Jane Smith', 11).valid? #=> false
User.new(nil, 30).valid?          #=> false
```

Leggi ActiveModel online: https://riptutorial.com/it/ruby-on-rails/topic/1773/activemodel

# Capitolo 7: ActiveRecord

## Examples

### Creazione manuale di un modello

L'utilizzo di scaffolding è rapido e semplice se si è nuovi a Rails o si sta creando una nuova applicazione, in seguito può essere utile solo eseguirlo da solo per evitare la necessità di passare attraverso il codice generato dallo scaffold per ridurlo. (rimuovere le parti non utilizzate, ecc.).

La creazione di un modello può essere semplice come creare un file in `app/models` .

Il modello più semplice, in `ActiveRecord` , è una classe che estende `ActiveRecord::Base` .

```
class User < ActiveRecord::Base
end
```

I file di modello sono memorizzati in `app/models/` e il nome del file corrisponde al nome singolare della classe:

```
# user
app/models/user.rb

# SomeModel
app/models/some_model.rb
```

La classe erediterà tutte le funzionalità di ActiveRecord: metodi di interrogazione, convalide, callback, ecc.

```
# Searches the User with ID 1
User.find(1)
```

Nota: assicurarsi che la tabella per il modello corrispondente esista. In caso contrario, è possibile creare la tabella creando una migrazione

È possibile generare un modello ed è la migrazione per terminale dal seguente comando

```
rails g model column_name1:data_type1, column_name2:data_type2, ...
```

e può anche assegnare una chiave esterna (relazione) al modello seguendo il comando

```
rails g model column_name:data_type, model_name:references
```

### Creazione di un modello tramite generatore

Ruby on Rails fornisce un generatore di `model` che è possibile utilizzare per creare modelli ActiveRecord. Basta usare i `rails generate model` e fornire il nome del modello.

```
$ rails g model user
```

Oltre al file di modello in `app/models` , il generatore creerà anche:

- il test in `test/models/user_test.rb`
- le Fixtures in `test/fixtures/users.yml`
- la migrazione del database in `db/migrate/XXX_create_users.rb`

È anche possibile generare alcuni campi per il modello quando lo si genera.

```
$ rails g model user email:string sign_in_count:integer birthday:date
```

Questo creerà le colonne email, sign_in_count e birthday nel tuo database, con i tipi appropriati.

### Creazione di una migrazione

# Aggiungi / rimuovi campi nelle tabelle esistenti

Crea una migrazione eseguendo:

```
rails generate migration AddTitleToCategories title:string
```

Ciò creerà una migrazione che aggiunge una colonna del `title` a una tabella delle `categories` :

```
class AddTitleToCategories < ActiveRecord::Migration[5.0]
  def change
    add_column :categories, :title, :string
  end
end
```

Allo stesso modo, è possibile generare una migrazione per rimuovere una colonna: le `rails generate migration RemoveTitleFromCategories title:string`

Ciò creerà una migrazione che rimuove una colonna del `title` dalla tabella delle `categories` :

```
class RemoveTitleFromCategories < ActiveRecord::Migration[5.0]
  def change
    remove_column :categories, :title, :string
  end
end
```

Mentre, in senso stretto, **non** è **necessario** specificare il **tipo** ( `:string` in questo caso) per rimuovere una colonna, **è utile** , poiché fornisce le informazioni necessarie per **il rollback** .

# Crea un tavolo

Crea una migrazione eseguendo:

```
rails g CreateUsers name bio
```

Rails riconosce l'intenzione di creare una tabella dal prefisso `Create` , il resto del nome della migrazione verrà utilizzato come nome di tabella. L'esempio fornito genera quanto segue:

```
class CreateUsers < ActiveRecord::Migration
  def change
    create_table :users do |t|
      t.string :name
      t.string :bio
    end
  end
end
```

Si noti che il comando di creazione non ha specificato tipi di colonne e `string` stata utilizzata la `string` predefinita.

# Crea una tabella di join

Crea una migrazione eseguendo:

```
rails g CreateJoinTableParticipation user:references group:references
```

Rails rileva l'intento di creare una tabella di join trovando `JoinTable` nel nome della migrazione. Tutto il resto è determinato dai nomi dei campi che dai dopo il nome.

```
class CreateJoinTableParticipation < ActiveRecord::Migration
  def change
    create_join_table :users, :groups do |t|
      # t.index [:user_id, :group_id]
      # t.index [:group_id, :user_id]
    end
  end
end
```

Decommentare le dichiarazioni `index` necessarie e cancellare il resto.

# Precedenza

Si noti che il nome di migrazione di esempio `CreateJoinTableParticipation` corrisponde alla regola per la creazione della tabella: ha un prefisso `Create` . Ma non ha generato un semplice `create_table` . Questo perché il generatore di migrazione ( codice sorgente ) utilizza una **prima corrispondenza** dell'elenco seguente:

- `(Add|Remove)<ignored>(To|From)<table_name>`
- `<ignored>JoinTable<ignored>`

- `Create<table_name>`

## Introduzione ai callback

Un callback è un metodo che viene richiamato in momenti specifici del ciclo di vita di un oggetto (subito prima o dopo la creazione, la cancellazione, l'aggiornamento, la convalida, il salvataggio o il caricamento dal database).

Ad esempio, supponiamo di avere un elenco che scade entro 30 giorni dalla creazione.

Un modo per farlo è così:

```
class Listing < ApplicationRecord
  after_create :set_expiry_date

  private

  def set_expiry_date
    expiry_date = Date.today + 30.days
    self.update_column(:expires_on, expiry_date)
  end
end
```

Tutti i metodi disponibili per le richiamate sono i seguenti, nello stesso ordine in cui vengono chiamati durante l'operazione di ciascun oggetto:

**Creare un oggetto**

- before_validation
- after_validation
- before_save
- around_save
- before_create
- around_create
- after_create
- after_save
- after_commit / after_rollback

**Aggiornamento di un oggetto**

- before_validation
- after_validation
- before_save
- around_save
- before_update
- around_update
- after_update
- after_save
- after_commit / after_rollback

**Distruggere un oggetto**

- before_destroy
- around_destroy
- after_destroy
- after_commit / after_rollback

***NOTA:*** *after_save viene eseguito sia su create che su update, ma sempre dopo i callback più specifici after_create e after_update, indipendentemente dall'ordine in cui sono state eseguite le macro chiamate.*

## Creare una tabella di join utilizzando le migrazioni

Particolarmente utile per la relazione `has_and_belongs_to_many` , è possibile creare manualmente una tabella di join utilizzando il metodo `create_table` . Supponiamo che tu abbia due modelli `Tags` e `Proyects` e desideri associarli usando una relazione `has_and_belongs_to_many` . È necessaria una tabella di join per associare le istanze di entrambe le classi.

```
class CreateProjectsTagsJoinTableMigration < ActiveRecord::Migration
  def change
    create_table :projects_tags, id: false do |t|
      t.integer :project_id
      t.integer :tag_id
    end
  end
end
```

Il nome effettivo della tabella deve seguire questa convenzione: il modello che precede alfabeticamente l'altro deve prima passare. **P** roject precede **T** ags quindi il nome della tabella è project_tags.

Inoltre, poiché lo scopo di questa tabella è quello di instradare l'associazione tra le istanze di due modelli, l'ID effettivo di ogni record in questa tabella non è necessario. Si specifica questo passando `id: false`

Infine, come è convenzione in Rails, il nome della tabella deve essere la forma plurale composta dei singoli modelli, ma la colonna della tabella deve essere in forma singolare.

## Testare manualmente i tuoi modelli

Testare i modelli di record attivi tramite l'interfaccia della riga di comando è semplice. Passare alla directory dell'app nel terminale e digitare nella `rails console` per avviare la console di Rails. Da qui, puoi eseguire metodi di registrazione attivi sul tuo database.

Ad esempio, se si dispone di uno schema di database con una tabella Utenti con un `name:string` colonna di `name:string` e `email:string` , è possibile eseguire:

```
User.create name: "John", email: "john@example.com"
```

Quindi, per mostrare quel record, puoi eseguire:

```
User.find_by email: "john@example.com"
```

O se questo è il tuo primo o unico record, puoi semplicemente ottenere il primo record eseguendo:

```
User.first
```

### Utilizzando un'istanza di modello per aggiornare una riga

Supponiamo tu abbia un modello `User`

```
class User < ActiveRecord::Base
end
```

Ora per aggiornare il `first_name` e il `last_name` di un utente con `id = 1` , puoi scrivere il seguente codice.

```
user = User.find(1)
user.update(first_name: 'Kashif', last_name: 'Liaqat')
```

La chiamata `update` tenterà di aggiornare gli attributi dati in una singola transazione, restituendo `true` se ha esito positivo e `false` caso contrario.

Leggi ActiveRecord online: https://riptutorial.com/it/ruby-on-rails/topic/828/activerecord

# Capitolo 8: ActiveRecord Associations

## Examples

### appartiene a

A `belongs_to` association imposta una connessione one-to-one con un altro modello, quindi ogni istanza del modello dichiarante "appartiene a" un'istanza dell'altro modello.

Ad esempio, se l'applicazione include utenti e post e ogni post può essere assegnato esattamente a un utente, dichiarerai il modello di post in questo modo:

```
class Post < ApplicationRecord
  belongs_to :user
end
```

Nella tua struttura tabella potresti avere

```
create_table "posts", force: :cascade do |t|
  t.integer  "user_id",  limit: 4
end
```

### Ha uno

`has_one` imposta una connessione one-to-one con un altro modello, ma con semantica diversa. Questa associazione indica che ogni istanza di un modello contiene o possiede un'istanza di un altro modello.

Ad esempio, se ogni utente nella tua applicazione ha un solo account, dichiareresti il modello utente in questo modo:

```
class User < ApplicationRecord
  has_one :account
end
```

In Active Record, quando si ha una relazione `has_one` , il record attivo garantisce che l'unico record esista con la chiave esterna.

Qui nel nostro esempio: nella tabella degli account, può esserci un solo record con un id utente particolare. Se si tenta di associare un altro account per lo stesso utente, rende la chiave esterna della voce precedente come null (rendendola orfana) e ne crea automaticamente una nuova. Rende nullo la voce precedente anche se il salvataggio fallisce perché la nuova voce mantenga la coerenza.

```
user = User.first
user.build_account(name: "sample")
user.save   [Saves it successfully, and creates an entry in accounts table with user_id 1]
```

```
user.build_account(name: "sample1")  [automatically makes the previous entry's foreign key
null]
user.save  [creates the new account with name sample 1 and user_id 1]
```

## ha molti

`has_many` indica una connessione uno-a-molti con un altro modello. Questa associazione generalmente si trova sull'altro lato di un'associazione_part_di.

Questa associazione indica che ogni istanza del modello ha zero o più istanze di un altro modello.

Ad esempio, in un'applicazione contenente utenti e post, il modello utente potrebbe essere dichiarato in questo modo:

```
class User < ApplicationRecord
  has_many :posts
end
```

La struttura della tabella di `Post` rimarrebbe la stessa come nel `belongs_to` esempio; al contrario, l' `User` non richiederebbe alcuna modifica dello schema.

Se si desidera ottenere l'elenco di tutti i post pubblicati per l' `User` , è possibile aggiungere quanto segue (ovvero è possibile aggiungere ambiti agli oggetti di associazione):

```
class User < ApplicationRecord
  has_many :published_posts, -> { where("posts.published IS TRUE") }, class_name: "Post"
end
```

## Associazione polimorfica

Questo tipo di associazione consente ad un modello di ActiveRecord di appartenere a più di un tipo di record del modello. Esempio comune:

```
class Human < ActiveRecord::Base
  has_one :address, :as => :addressable
end

class Company < ActiveRecord::Base
  has_one :address, :as => :addressable
end

class Address < ActiveRecord::Base
  belongs_to :addressable, :polymorphic => true
end
```

Senza questa associazione, avresti tutte queste chiavi esterne nella tua tabella Indirizzi ma solo tu avresti mai un valore per una di esse perché un indirizzo, in questo scenario, può appartenere solo a una entità (Umana o Azienda). Ecco come apparirà:

```
class Address < ActiveRecord::Base
  belongs_to :human
```

```
  belongs_to :company
end
```

## L'has_many: attraverso l'associazione

A `has_many :through` associazione viene spesso utilizzato per impostare una connessione `many-to-many` con un altro modello. Questa associazione indica che il modello di dichiarazione può essere abbinato a zero o più istanze di un altro modello procedendo attraverso un terzo modello.

Ad esempio, si consideri una pratica medica in cui i pazienti fanno appuntamenti per vedere i medici. Le dichiarazioni di associazione pertinenti potrebbero assomigliare a questo:

```
class Physician < ApplicationRecord
  has_many :appointments
  has_many :patients, through: :appointments
end

class Appointment < ApplicationRecord
  belongs_to :physician
  belongs_to :patient
end

class Patient < ApplicationRecord
  has_many :appointments
  has_many :physicians, through: :appointments
end
```

## Has_one: attraverso l'associazione

A `has_one :through` associazione imposta una connessione `one-to-one` con un altro modello. Questa associazione indica che il modello di dichiarazione può essere abbinato a un'istanza di un altro modello procedendo attraverso un terzo modello.

Ad esempio, se ogni `supplier` ha un `account` e ogni account è associato a una cronologia dell'account, il modello del fornitore potrebbe essere simile al seguente:

```
class Supplier < ApplicationRecord
  has_one :account
  has_one :account_history, through: :account
end

class Account < ApplicationRecord
  belongs_to :supplier
  has_one :account_history
end

class AccountHistory < ApplicationRecord
  belongs_to :account
end
```

## L'associazione has_and_belongs_to_many

`has_and_belongs_to_many` crea una connessione diretta `many-to-many` con un altro modello, senza

alcun modello `has_and_belongs_to_many` .

Ad esempio, se l'applicazione include `assemblies` e `parts` , con ciascun assieme con molte parti e ciascuna parte che appare in molti assiemi, è possibile dichiarare i modelli in questo modo:

```
class Assembly < ApplicationRecord
  has_and_belongs_to_many :parts
end

class Part < ApplicationRecord
  has_and_belongs_to_many :assemblies
end
```

## Associazione autoreferenziale

L'associazione autoreferenziale viene utilizzata per associare un modello a se stesso. L'esempio più frequente sarebbe, per gestire l'associazione tra un amico e il suo seguace.

ex.

```
rails g model friendship user_id:references friend_id:integer
```

ora puoi associare modelli come;

```
class User < ActiveRecord::Base
  has_many :friendships
  has_many :friends, :through => :friendships
  has_many :inverse_friendships, :class_name => "Friendship", :foreign_key => "friend_id"
  has_many :inverse_friends, :through => :inverse_friendships, :source => :user
end
```

e l'altro modello sarà simile;

```
class Friendship < ActiveRecord::Base
  belongs_to :user
  belongs_to :friend, :class_name => "User"
end
```

Leggi ActiveRecord Associations online: https://riptutorial.com/it/ruby-on-rails/topic/1820/activerecord-associations

# Capitolo 9: ActiveSupport

## Osservazioni

ActiveSupport è una utility gemma di strumenti generici utilizzati dal resto del framework Rails.

Uno dei principali modi in cui fornisce questi strumenti consiste nel tenere traccia dei tipi nativi di Ruby. Questi sono indicati come le **estensioni del nucleo** .

## Examples

### Estensioni principali: accesso con stringhe

# String # a

Restituisce una sottostringa di un oggetto stringa. Stessa interfaccia di `String#[]` .

```
str = "hello"
str.at(0)       # => "h"
str.at(1..3)    # => "ell"
str.at(-2)      # => "l"
str.at(-2..-1) # => "lo"
str.at(5)       # => nil
str.at(5..-1)   # => ""
```

# String # da

Restituisce una sottostringa dalla posizione data alla fine della stringa.

```
str = "hello"
str.from(0)  # => "hello"
str.from(3)  # => "lo"
str.from(-2) # => "lo"
```

# String # per

Restituisce una sottostringa dall'inizio della stringa alla posizione specificata.
Se la posizione è negativa, viene contata dalla fine della stringa.

```
str = "hello"
str.to(0)  # => "h"
str.to(3)  # => "hell"
str.to(-2) # => "hell"
```

`from` e `to` poter essere usato in tandem.

```
str = "hello"
str.from(0).to(-1) # => "hello"
str.from(1).to(-2) # => "ell"
```

# String # primo

Restituisce il primo carattere o un determinato numero di caratteri fino alla lunghezza della stringa.

```
str = "hello"
str.first    # => "h"
str.first(1) # => "h"
str.first(2) # => "he"
str.first(0) # => ""
str.first(6) # => "hello"
```

# String # ultima

Restituisce l'ultimo carattere o un determinato numero di caratteri dalla fine della stringa contando all'indietro.

```
str = "hello"
str.last    # => "o"
str.last(1) # => "o"
str.last(2) # => "lo"
str.last(0) # => ""
str.last(6) # => "hello"
```

### Estensioni principali: conversione da stringa a data / ora

# String # to_time

Converte una stringa in un valore Time. Il parametro del `form` può essere `:utc` o `:local` , il valore predefinito è `:local` .

```
"13-12-2012".to_time               # => 2012-12-13 00:00:00 +0100
"06:12".to_time                    # => 2012-12-13 06:12:00 +0100
"2012-12-13 06:12".to_time         # => 2012-12-13 06:12:00 +0100
"2012-12-13T06:12".to_time         # => 2012-12-13 06:12:00 +0100
"2012-12-13T06:12".to_time(:utc)   # => 2012-12-13 06:12:00 UTC
"12/13/2012".to_time               # => ArgumentError: argument out of range
```

# String # to_date

Converte una stringa in un valore Date.

```
"1-1-2012".to_date   # => Sun, 01 Jan 2012
"01/01/2012".to_date # => Sun, 01 Jan 2012
"2012-12-13".to_date # => Thu, 13 Dec 2012
"12/13/2012".to_date # => ArgumentError: invalid date
```

# String # to_datetime

Converte una stringa in un valore DateTime.

```
"1-1-2012".to_datetime             # => Sun, 01 Jan 2012 00:00:00 +0000
"01/01/2012 23:59:59".to_datetime # => Sun, 01 Jan 2012 23:59:59 +0000
"2012-12-13 12:50".to_datetime     # => Thu, 13 Dec 2012 12:50:00 +0000
"12/13/2012".to_datetime           # => ArgumentError: invalid date
```

## Estensioni principali: esclusione dalla stringa

# String # escludono?

L'inverso di `String#include?`

```
"hello".exclude? "lo" # => false
"hello".exclude? "ol" # => true
"hello".exclude? ?h   # => false
```

## Estensioni principali: filtri stringa

# String # squish

Restituisce una versione della stringa specificata senza spazi vuoti iniziali o finali e combina tutti gli spazi bianchi consecutivi all'interno in spazi singoli. Squish versione distruttiva `squish!` opera direttamente sull'istanza della stringa.

Gestisce sia gli spazi bianchi ASCII che Unicode.

```
%{ Multi-line
   string }.squish                   # => "Multi-line string"
" foo   bar    \n   \t   boo".squish # => "foo bar boo"
```

# String # remove

Restituisce una nuova stringa con tutte le occorrenze dei modelli rimossi. La versione distruttiva `remove!` opera direttamente sulla stringa data.

```
str = "foo bar test"
str.remove(" test")                 # => "foo bar"
str.remove(" test", /bar/)          # => "foo "
```

# String # troncare

Restituisce una copia di una data stringa troncata ad una lunghezza specifica se la stringa è più lunga della lunghezza.

```
'Once upon a time in a world far far away'.truncate(27)
# => "Once upon a time in a wo..."
```

Passa una stringa o regexp `:separator` per troncare in una naturale interruzione

```
'Once upon a time in a world far far away'.truncate(27, separator: ' ')
# => "Once upon a time in a..."

'Once upon a time in a world far far away'.truncate(27, separator: /\s/)
# => "Once upon a time in a..."
```

# String # truncate_words

Restituisce una stringa troncata dopo un determinato numero di parole.

```
'Once upon a time in a world far far away'.truncate_words(4)
# => "Once upon a time..."
```

Passa una stringa o regexp per specificare un diverso separatore di parole

```
'Once<br>upon<br>a<br>time<br>in<br>a<br>world'.truncate_words(5, separator: '<br>')
# => "Once<br>upon<br>a<br>time<br>in..."
```

Gli ultimi caratteri saranno sostituiti con `:omission` string (predefinito su "...")

```
'And they found that many people were sleeping better.'.truncate_words(5, omission: '...
(continued)')
# => "And they found that many... (continued)"
```

# String # strip_heredoc

Indentazione di strisce in heredocs. Cerca la linea non vuota meno rientrata e rimuove quella quantità di spazi bianchi iniziali.

```
if options[:usage]
  puts <<-USAGE.strip_heredoc
```

```
    This command does such and such.

    Supported options are:
      -h         This message
      ...
  USAGE
end
```

l'utente vedrebbe

```
This command does such and such.

Supported options are:
-h         This message
...
```

### Core Extensions: String Inflection

# String # plurale

Ritorni della forma plurale della stringa. Facoltativamente, accetta un parametro di `count` e restituisce una forma singolare se `count == 1` . Accetta anche un parametro `locale` per la pluralizzazione specifica della lingua.

```
'post'.pluralize             # => "posts"
'octopus'.pluralize          # => "octopi"
'sheep'.pluralize            # => "sheep"
'words'.pluralize            # => "words"
'the blue mailman'.pluralize # => "the blue mailmen"
'CamelOctopus'.pluralize     # => "CamelOctopi"
'apple'.pluralize(1)         # => "apple"
'apple'.pluralize(2)         # => "apples"
'ley'.pluralize(:es)         # => "leyes"
'ley'.pluralize(1, :es)      # => "ley"
```

# String # singolarizzare

Restituisce la forma singolare della stringa. Accetta un parametro `locale` opzionale.

```
'posts'.singularize            # => "post"
'octopi'.singularize           # => "octopus"
'sheep'.singularize            # => "sheep"
'word'.singularize             # => "word"
'the blue mailmen'.singularize # => "the blue mailman"
'CamelOctopi'.singularize      # => "CamelOctopus"
'leyes'.singularize(:es)       # => "ley"
```

# String # constantize

Cerca di trovare una costante dichiarata con il nome specificato nella stringa. Solleva un `NameError` quando il nome non è in CamelCase o non è inizializzato.

```
'Module'.constantize  # => Module
'Class'.constantize   # => Class
'blargle'.constantize # => NameError: wrong constant name blargle
```

# String # safe_constantize

Esegue una `constantize` ma restituisce `nil` invece di sollevare `NameError` .

```
'Module'.safe_constantize  # => Module
'Class'.safe_constantize   # => Class
'blargle'.safe_constantize # => nil
```

# String # camelize

Converte le stringhe in UpperCamelCase per impostazione predefinita, se `:lower` viene dato come param converte in lowerCamelCase.

alias: `camelcase`

**Nota:** sarà anche convertire `/` a `::` che è utile per la conversione di percorsi per gli spazi dei nomi.

```
'active_record'.camelize                # => "ActiveRecord"
'active_record'.camelize(:lower)        # => "activeRecord"
'active_record/errors'.camelize         # => "ActiveRecord::Errors"
'active_record/errors'.camelize(:lower) # => "activeRecord::Errors"
```

# String # titleize

Capitalizza tutte le parole e sostituisce alcuni caratteri nella stringa per creare un titolo dall'aspetto più gradevole.

alias: `titlecase`

```
'man from the boondocks'.titleize # => "Man From The Boondocks"
'x-men: the last stand'.titleize  # => "X Men: The Last Stand"
```

# String # sottolineano

Crea una forma in lettere minuscole sottolineata dall'espressione nella stringa. Il contrario di `camelize` .

**Nota: il carattere di** `underscore` cambierà anche `::` in `/` per convertire spazi dei nomi in tracciati.

```
'ActiveModel'.underscore         # => "active_model"
'ActiveModel::Errors'.underscore # => "active_model/errors"
```

# String # dasherize

Sostituisce i trattini bassi con trattini nella stringa.

```
'puni_puni'.dasherize # => "puni-puni"
```

# String # demodulize

Rimuove la parte del modulo dall'espressione costante nella stringa.

```
'ActiveRecord::CoreExtensions::String::Inflections'.demodulize # => "Inflections"
'Inflections'.demodulize                                       # => "Inflections"
'::Inflections'.demodulize                                     # => "Inflections"
''.demodulize                                                  # => ''
```

# String # deconstantize

Rimuove il segmento più a destra dall'espressione costante nella stringa.

```
'Net::HTTP'.deconstantize   # => "Net"
'::Net::HTTP'.deconstantize # => "::Net"
'String'.deconstantize      # => ""
'::String'.deconstantize    # => ""
''.deconstantize            # => ""
```

# String # Parametrizzazione

Sostituisce caratteri speciali in una stringa in modo che possa essere utilizzato come parte di un URL "carino".

```
"Donald E. Knuth".parameterize # => "donald-e-knuth"
```

Conserva il caso dei caratteri in una stringa con l'argomento `:preserve_case` .

```
"Donald E. Knuth".parameterize(preserve_case: true) # => "Donald-E-Knuth"
```

Un caso d'uso molto comune per `parameterize` è sovrascrivere il metodo `to_param` di un modello ActiveRecord per supportare più slug url descrittivi.

```
class Person < ActiveRecord::Base
  def to_param
    "#{id}-#{name.parameterize}"
  end
end

Person.find(1).to_param # => "1-donald-e-knuth"
```

# String # tableize

Crea il nome di una tabella come fa Rails per i modelli ai nomi delle tabelle. Pluralizza l'ultima parola nella stringa.

```
'RawScaledScorer'.tableize # => "raw_scaled_scorers"
'ham_and_egg'.tableize     # => "ham_and_eggs"
'fancyCategory'.tableize   # => "fancy_categories"
```

# String # classificano

Restituisce una stringa di nome di classe da un nome di tabella plurale come Rails per i nomi di tabelle per i modelli.

```
'ham_and_eggs'.classify # => "HamAndEgg"
'posts'.classify        # => "Post"
```

# String # umanizzare

`_id` maiuscolo la prima parola, trasforma i caratteri di sottolineatura in spazi e toglie uno `_id` finale se presente.

```
'employee_salary'.humanize              # => "Employee salary"
'author_id'.humanize                    # => "Author"
'author_id'.humanize(capitalize: false) # => "author"
'_id'.humanize                          # => "Id"
```

# String # upcase_first

Converte solo il primo carattere in maiuscolo.

```
'what a Lovely Day'.upcase_first # => "What a Lovely Day"
'w'.upcase_first                 # => "W"
''.upcase_first                  # => ""
```

# String # foreign_key

Crea un nome di chiave esterna da un nome di classe. Passa `false` param per disabilitare l'aggiunta di _ tra nome ed `id` .

```
'Message'.foreign_key        # => "message_id"
'Message'.foreign_key(false) # => "messageid"
'Admin::Post'.foreign_key    # => "post_id"
```

Leggi ActiveSupport online: https://riptutorial.com/it/ruby-on-rails/topic/4490/activesupport

# Capitolo 10: Aggiornamento di Rails

## Examples

### Aggiornamento da Rails 4.2 a Rails 5.0

*Nota: prima di aggiornare l'app Rails, assicurati sempre di salvare il codice su un sistema di controllo versione, come Git.*

---

Per eseguire l'aggiornamento da Rails 4.2 a Rails 5.0, è necessario utilizzare Ruby 2.2.2 o versione successiva. Dopo aver aggiornato la tua versione di Ruby, se necessario, vai al tuo Gemfile e cambia linea:

```
gem 'rails', '4.2.X'
```

a:

```
gem 'rails', '~> 5.0.0'
```

e sulla riga di comando eseguire:

```
$ bundle update
```

Ora esegui l'attività di aggiornamento usando il comando:

```
$ rake rails:update
```

Questo ti aiuterà ad aggiornare i file di configurazione. Ti verrà richiesto di sovrascrivere i file e hai diverse opzioni da inserire:

- Sì, sì, sovrascrivi
- n - no, non sovrascrivere
- a - tutto, sovrascrivi questo e tutti gli altri
- q - esci, abortisci
- d - diff, mostra le differenze tra il vecchio e il nuovo
- h - aiuto

In genere, è necessario verificare le differenze tra i file vecchi e quelli nuovi per assicurarsi che non vengano apportate modifiche indesiderate.

Rails 5.0 I modelli `ActiveRecord` ereditano da `ApplicationRecord` , piuttosto che `ActiveRecord::Base` . `ApplicationRecord` è la superclasse per tutti i modelli, simile a come `ApplicationController` è la superclasse per i controller. Per tenere conto di questo nuovo modo in cui i modelli sono gestiti, devi creare un file nella tua `app/models/` cartella chiamata `application_record.rb` e quindi modificare il contenuto di quel file in modo che sia:

```
class ApplicationRecord < ActiveRecord::Base
  self.abstract_class = true
end
```

Rails 5.0 gestisce anche i callback leggermente diversi. Le callback che restituiscono `false` non fermeranno la catena di callback, il che significa che i callback successivi continueranno a essere eseguiti, diversamente da Rails 4.2. Quando esegui l'upgrade, il comportamento di Rails 4.2 rimarrà, sebbene tu possa passare al comportamento di Rails 5.0 aggiungendo:

```
ActiveSupport.halt_callback_chains_on_return_false = false
```

nel file `config/application.rb` . È possibile interrompere esplicitamente la catena di callback chiamando `throw(:abort)` .

In Rails 5.0, `ActiveJob` erediterà da `ApplicationJob` , piuttosto che `ActiveJob::Base` come in Rails 4.2. Per eseguire l'aggiornamento a Rails 5.0, creare un file chiamato `application_job.rb` nella cartella `app/jobs/` . Modifica i contenuti di quel file come:

```
class ApplicationJob < ActiveJob::Base
end
```

Quindi, è necessario modificare tutti i processi in modo che `ActiveJob::Base` ereditati da `ApplicationJob` anziché `ActiveJob::Base` .

Uno degli altri più grandi cambiamenti di Rails 5.0 non richiede modifiche al codice, ma cambierà il modo in cui usi la riga di comando con le tue app Rails. Sarai in grado di utilizzare `bin/rails` o solo `rails` per eseguire attività e test. Ad esempio, invece di utilizzare `$ rake db:migrate` , ora puoi eseguire `$ rails db:migrate` . Se si esegue `$ bin/rails` , è possibile visualizzare tutti i comandi disponibili. Nota che molte delle attività che ora possono essere eseguite con `bin/rails` funzionano ancora utilizzando `rake` .

Leggi Aggiornamento di Rails online: https://riptutorial.com/it/ruby-on-rails/topic/3496/aggiornamento-di-rails

# Capitolo 11: Aggiungi il pannello di amministrazione

## introduzione

Se si desidera aggiungere un pannello di amministrazione all'applicazione rota, è solo questione di minuti.

## Sintassi

1. Apri gem file e writer gem 'rails_admin', '~> 1.0'
2. installazione bundle
3. rails g rails_admin: installa
4. ti chiederà il percorso di amministrazione se vuoi andare con il predefinito premi Invio.
5. Ora vai su app / config / initializers / rails_admin.rb e incolla questo codice: config.authorize_with fai redirect_to main_app.root_path a meno che current_user.try (: admin?) Termini Questo codice consentirà solo agli utenti admin di accedere a yoursite.com/admin other gli utenti verranno reindirizzati al rootpath.
6. Per maggiori dettagli controlla la documentazione di questa gemma. https://github.com/sferik/rails_admin/wiki

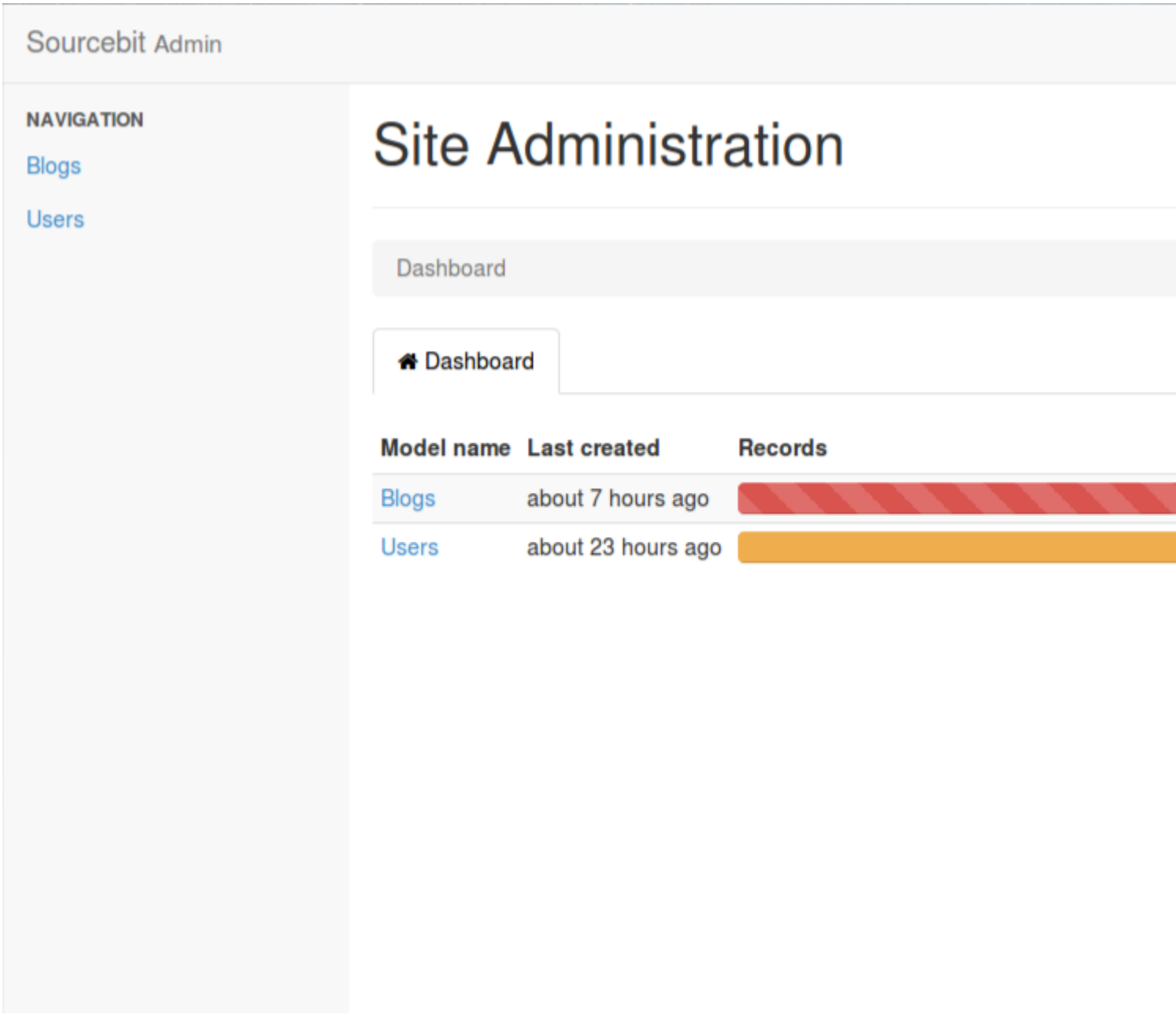
## Osservazioni

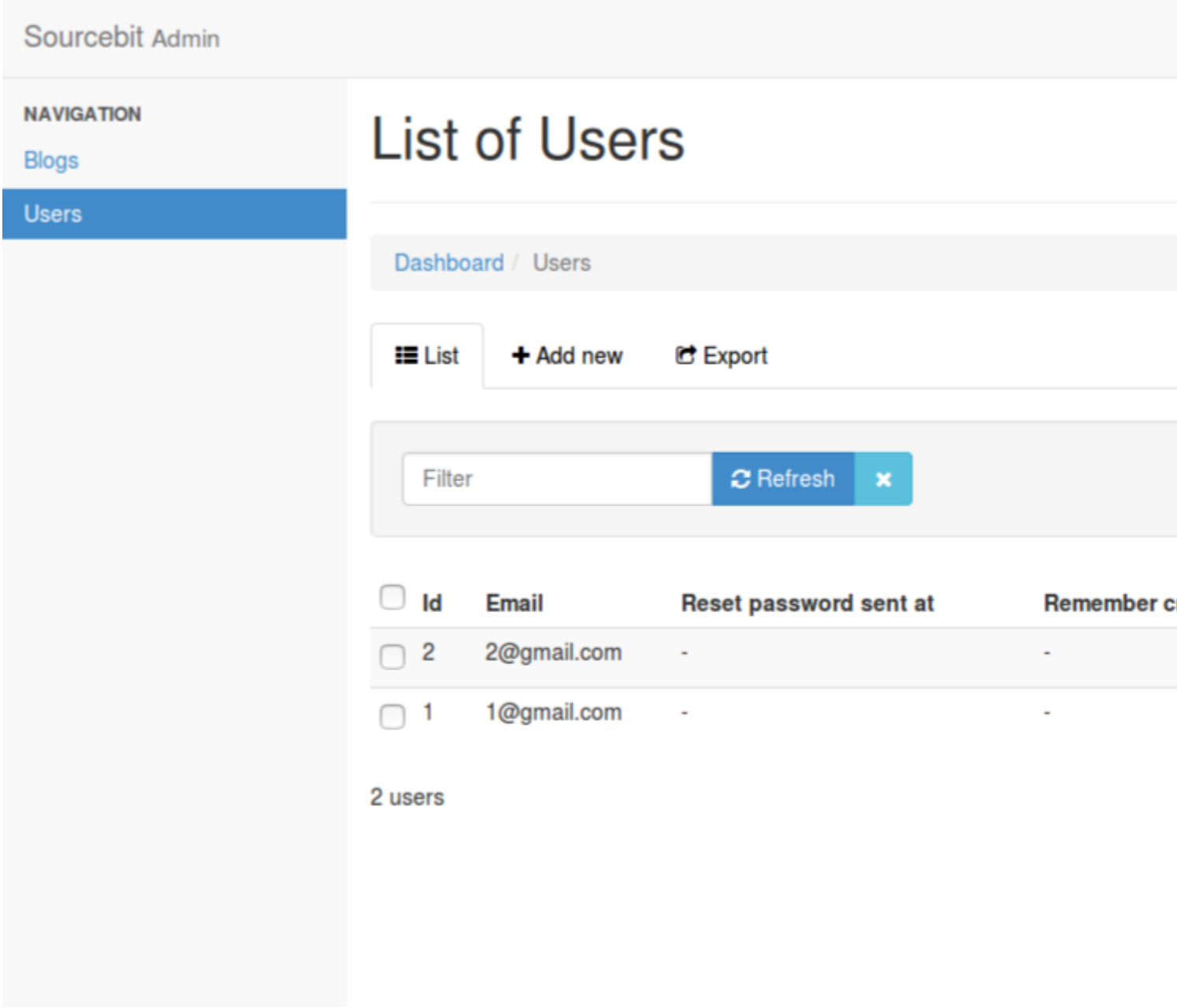
Usalo se vuoi avere Admin sul tuo sito web altrimenti non ce n'è bisogno. È più facile e potente della gemma active_admin. Puoi aggiungerlo in qualsiasi momento dopo aver creato gli utenti e non dimenticare di rendere qualsiasi amministratore utente prima del 4 ° passaggio. Usa cancan per assegnare ruoli.

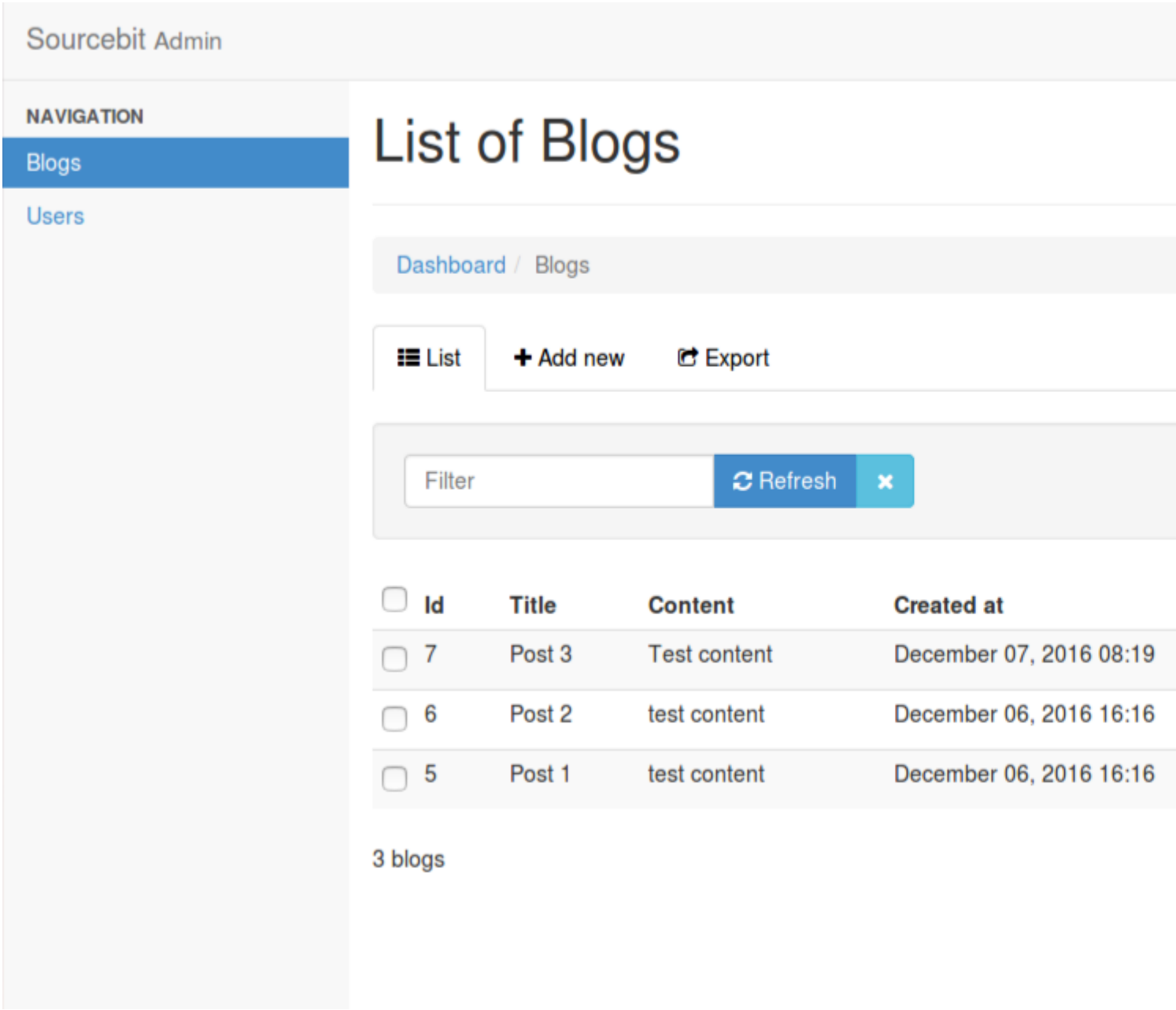
## Examples

### Quindi ecco alcune schermate dal pannello di amministrazione usando rails_admin gem.

Come puoi vedere il layout di questa gemma è molto accattivante e facile da usare.


Sourcebit Admin
NAVIGATION
Blogs
Users
Site Administration
Dashboard
Dashboard
Model name
Last created
Records
Blogs
about 7 hours ago
Users
about 23 hours ago


Sourcebit Admin
NAVIGATION
Blogs
Users
List of Users
Dashboard / Users
List
Add new
Export
Filter
Refresh
Id
Email
Reset password sent at
2
2@gmail.com
-
1
1@gmail.com
-
2 users




Leggi Aggiungi il pannello di amministrazione online: https://riptutorial.com/it/ruby-on-rails/topic/8128/aggiungi-il-pannello-di-amministrazione

# Capitolo 12: Aggiunta di un Amazon RDS all'applicazione rails

## introduzione

Passaggi per creare un'istanza RDS AWS e configurare il file database.yml installando i connettori richiesti.

## Examples

### Considera che stiamo collegando MYSQL RDS con la tua applicazione di binari.

**Passi per creare il database MYSQL**

1. Accedi all'account Amazon e seleziona il servizio RDS
2. Selezionare `Launch DB Instance` dalla scheda `Launch DB Instance`
3. Di DEFAUL MYSQL Community Edition verrà selezionato, quindi fare clic sul pulsante `select`
4. Seleziona lo scopo del database, pronuncia la `production` e fai clic sul `next step`
5. Fornire la `mysql version, storage size, DB Instance Identifier, Master Username and Password` e fare clic sul `next step`
6. Immettere il `Database Name` e fare clic su `Launch DB Instance`
7. Attendi fino a quando non viene creata l'istanza. Una volta che l'istanza viene creata, troverai un Endpoint, copia questo punto d'ingresso (che viene chiamato hostname)

**Installazione dei connettori**

Aggiungi l'adattatore del database MySQL al gemfile del tuo progetto,

```
gem 'mysql2'
```

Installa le tue gemme aggiunte,

```
bundle install
```

Alcuni altri adattatori di database sono,

- `gem 'pg'` per PostgreSQL
- `gem 'activerecord-oracle_enhanced-adapter'` per Oracle
- `gem 'sql_server'` per SQL Server

**Configura il file database.yml del tuo progetto** Apri il tuo **file** config / database.yml

```
production:
```

```
adapter: mysql2
encoding: utf8
database: <%= RDS_DB_NAME %>  # Which you have entered you creating database
username: <%= RDS_USERNAME %> # db master username
password: <%= RDS_PASSWORD %> # db master password
host: <%= RDS_HOSTNAME %>     # db instance entrypoint
port: <%= RDS_PORT %>         # db post. For MYSQL 3306
```

Leggi Aggiunta di un Amazon RDS all'applicazione rails online: https://riptutorial.com/it/ruby-on-rails/topic/10922/aggiunta-di-un-amazon-rds-all-applicazione-rails

# Capitolo 13: API Rails

## Examples

### Creazione di un'applicazione solo API

Per creare un'applicazione Rails che sarà un server API, puoi iniziare con un sottoinsieme più limitato di Rails in Rails 5.

Per generare una nuova app Rails API:

```
rails new my_api --api
```

Cosa fa `--api` è rimuovere la funzionalità che non è necessaria quando si `--api` un'API. Ciò include sessioni, cookie, risorse e tutto ciò che fa funzionare Rails su un browser.

Inoltre, configurerà i generatori in modo che non generino visualizzazioni, helper e risorse durante la generazione di una nuova risorsa.

Quando si confronta `ApplicationController` su un'app Web rispetto a un'app API, si noterà che la versione Web si estende da `ActionController::Base` , mentre la versione API si estende da `ActionController::API` , che include un sottoinsieme di funzionalità molto più piccolo.

Leggi API Rails online: https://riptutorial.com/it/ruby-on-rails/topic/4305/api-rails

# Capitolo 14: Asset Pipeline

## introduzione

La pipeline di asset fornisce un framework per concatenare e minimizzare o comprimere le risorse JavaScript e CSS. Aggiunge anche la possibilità di scrivere queste risorse in altre lingue e pre-processori come CoffeeScript, Sass e ERB. Permette di integrare automaticamente le risorse della tua applicazione con le risorse di altre gemme. Ad esempio, jquery-rails include una copia di jquery.js e abilita le funzionalità AJAX in Rails.

## Examples

### Rake tasks

Di default `sprockets-rails` viene fornito con le seguenti attività di rake:

- `assets:clean[keep]` : rimuove i vecchi asset compilati
- `assets:clobber` : rimuove le risorse compilate
- `assets:environment` : carica l'ambiente di compilazione delle risorse
- `assets:precompile` : compila tutte le risorse denominate in `config.assets.precompile`

### File e direttive manifest

Nel `assets` initalizer ( `config/initializers/assets.rb` ) ci sono alcuni file esplicitamente definiti per essere precompilati.

```
# Precompile additional assets.
# application.coffee, application.scss, and all non-JS/CSS in app/assets folder are already
added.
# Rails.application.config.assets.precompile += %w( search.js )
```

In questo esempio `application.coffee` e `application.scss` sono chiamati "file manifest". Questi file devono essere utilizzati per includere altre risorse JavaScript o CSS. Sono disponibili i seguenti comandi:

- `require <path>` : Il `require` funzioni direttive simili a Ruby proprio `require` . Fornisce un modo per dichiarare una dipendenza su un file nel percorso e garantisce che venga caricato solo una volta prima del file sorgente.
- `require_directory <path>` : richiede tutti i file all'interno di una singola directory. È simile al `path/*` poiché non segue le directory annidate.
- `require_tree <path>` : richiede tutti i file nidificati in una directory. Il suo equivalente globale è `path/**/*` .
- `require_self` : fa sì che il corpo del file corrente venga inserito prima di ogni successiva `require` direttive. Utile nei file CSS, in cui è comune che il file di indice contenga stili globali che devono essere definiti prima di caricare altre dipendenze.
- `stub <path>`

: rimuove un file dall'essere incluso

- `depend_on <path>` : consente di indicare una dipendenza su un file senza includerlo. Questo è usato per scopi di cache. Qualsiasi modifica apportata al file delle dipendenze invaliderà la cache del file sorgente.

Un file `application.scss` potrebbe avere il seguente aspetto:

```
/*
 *= require bootstrap
 *= require_directory .
 *= require_self
 */
```

Un altro esempio è il file `application.coffee` . Qui con inclusi `jquery` e `Turbolinks` :

```
#= require jquery2
#= require jquery_ujs
#= require turbolinks
#= require_tree .
```

Se non usi CoffeeScript, ma JavaScript semplice, la sintassi sarebbe:

```
//= require jquery2
//= require jquery_ujs
//= require turbolinks
//= require_tree .
```

## Uso di base

Esistono due modi fondamentali per utilizzare la pipeline degli asset:

1. Quando si esegue un server in modalità di sviluppo, esso pre-elabora e prepara automaticamente le risorse al volo.

2. In modalità produzione, probabilmente lo utilizzerai per pre-processare, versionizzare, comprimere e compilare le tue risorse. È possibile farlo eseguendo il seguente comando:

   ```
   bundle exec rake assets:precompile
   ```

Leggi Asset Pipeline online: https://riptutorial.com/it/ruby-on-rails/topic/3386/asset-pipeline

# Capitolo 15: Autenticate Api usando Devise

## introduzione

Devise è la soluzione di autenticazione per Rails. Prima di proseguire, vorrei aggiungere una breve nota sull'API. Quindi API non gestisce le sessioni (è stateless), ovvero una che fornisce una risposta dopo la richiesta, e quindi non richiede ulteriore attenzione, il che significa che non è richiesto alcun stato precedente o futuro affinché il sistema funzioni, quindi ogni volta che si richiede al server di passare i dettagli di autenticazione con tutte le API e indicare a Devise di non memorizzare i dettagli di autenticazione.

## Examples

### Iniziare

Quindi prima creeremo il progetto di rotaie e il dispositivo di installazione

creare un'applicazione per rotaie

```
rails new devise_example
```

ora aggiungi inventare alla lista delle gemme

> puoi trovare un file chiamato 'Gemfile' nella radice del progetto rails

Quindi eseguire l' `bundle install`

Successivamente, è necessario eseguire il generatore:

```
rails generate devise:install
```

Ora su console puoi trovare poche istruzioni seguirlo.

Genera il modello di sviluppo

```
rails generate devise MODEL
```

Quindi esegui `rake db:migrate`

Per maggiori dettagli vai su: Devise Gem

---

## Token di autenticazione

Il token di autenticazione viene utilizzato per autenticare un utente con un token univoco, quindi

prima di procedere con la logica prima è necessario aggiungere il campo `auth_token` a un modello di `auth_token`

Quindi,

```
rails g migration add_authentication_token_to_users

class AddAuthenticationTokenToUsers < ActiveRecord::Migration
  def change
    add_column :users, :auth_token, :string, default: ""
    add_index :users, :auth_token, unique: true
  end
end
```

Quindi esegui `rake db:migrate`

Ora siamo tutti pronti per fare l'autenticazione usando `auth_token`

In `app/controllers/application_controllers.rb`

Prima questa linea ad esso

```
respond_to :html, :json
```

questo aiuterà l'applicazione rails a rispondere sia con html che json

Poi

```
protect_from_forgery with: :null
```

cambierà questo `:null` dato che non abbiamo a che fare con le sessioni.

ora aggiungeremo il metodo di autenticazione in application_controller

Quindi, per impostazione predefinita, Devise utilizza l'e-mail come campo univoco che può anche utilizzare campi personalizzati, in questo caso verificheremo utilizzando user_email e auth_token.

```
  before_filter do
    user_email = params[:user_email].presence
    user       = user_email && User.find_by_email(user_email)

    if user && Devise.secure_compare(user.authentication_token, params[:auth_token])
      sign_in user, store: false
    end
  end
```

> Nota: il codice sopra riportato si basa esclusivamente sulla tua logica, sto solo cercando di spiegare l'esempio di lavoro

Alla riga 6 del codice precedente puoi vedere che ho impostato `store: false` che impedirà di creare una sessione su ogni richiesta, quindi abbiamo ottenuto lo stateless

Leggi Autenticate Api usando Devise online: https://riptutorial.com/it/ruby-on-rails/topic/9787/autenticate-api-usando-devise

# Capitolo 16: Autenticazione API Rails 5

## Examples

### Autenticazione con Rails authenticate_with_http_token

```
authenticate_with_http_token do |token, options|
  @user = User.find_by(auth_token: token)
end
```

Puoi testare questo endpoint con `curl` facendo una richiesta come

```
curl -IH "Authorization: Token token=my-token" http://localhost:3000
```

Leggi Autenticazione API Rails 5 online: https://riptutorial.com/it/ruby-on-rails/topic/7852/autenticazione-api-rails-5

# Capitolo 17: Autenticazione utente in Rails

## introduzione

Devise è una gemma molto potente, ti consente di iscriverti, accedere e uscire dalle opzioni subito dopo l'installazione. Inoltre, l'utente può aggiungere autenticazioni e restrizioni alle sue applicazioni. Devise anche venire con le proprie opinioni, se l'utente vuole usare. Un utente può inoltre personalizzare la registrazione e accedere ai moduli in base alle proprie necessità e necessità. Va notato che Devise ti consiglia di implementare il tuo login se sei nuovo alle guide.

## Osservazioni

Al momento della generazione delle configurazioni di configurazione che utilizzano i `rails generate devise:install` , devise elencherà una serie di istruzioni sul terminale da seguire.

Se hai già un modello `USER` , eseguendo questo comando i `rails generate devise USER` aggiungerà le colonne necessarie al tuo modello `USER` esistente.

Usa questo metodo di supporto `before_action :authenticate_user!` nella parte superiore del controller per verificare se l' `user` è connesso o meno. in caso contrario, verranno reindirizzati alla pagina di accesso.

## Examples

### Autenticazione usando Devise

Aggiungi gemma al Gemfile:

```
gem 'devise'
```

Quindi eseguire il comando di `bundle install` del `bundle install` .

Usa il comando `$ rails generate devise:install` per generare il file di configurazione richiesto.

Imposta le opzioni URL predefinite per il mailer Devise in ogni ambiente Nell'ambiente di sviluppo aggiungi questa riga:

```
config.action_mailer.default_url_options = { host: 'localhost', port: 3000 }
```

alla tua `config/environments/development.rb`

allo stesso modo in produzione questo file di `config/environments/production.rb` e aggiungi

```
config.action_mailer.default_url_options = { host: 'your-site-url'}
```

Quindi creare un modello utilizzando: `$ rails generate devise USER` Dove `USER` è il nome della

classe per il quale si desidera implementare l'autenticazione.

Infine, esegui: `rake db:migrate` e siete tutti impostati.

## Viste personalizzate

Se è necessario configurare le viste, è possibile utilizzare i `rails generate devise:views` generatore di `rails generate devise:views` che copierà tutte le viste all'applicazione. Quindi puoi modificarli come desiderato.

Se si dispone di più di un modello Devise nella propria applicazione (ad esempio Utente e Amministratore), si noterà che Devise utilizza le stesse viste per tutti i modelli. Devise offre un modo semplice per personalizzare le visualizzazioni. `config.scoped_views = true` nel file `config/initializers/devise.rb` .

È anche possibile utilizzare il generatore per creare viste con scope: i `rails generate devise:views users`

Se si desidera generare solo alcune serie di viste, come quelle per il modulo registrabile e confermabile, utilizzare il flag -v: `rails generate devise:views -v registrations confirmations`

### Definire i filtri e gli helper del controller

Per configurare un controller con l'autenticazione utente usando devise, aggiungi questo before_action: (assumendo che il tuo modello di dispositivo sia 'Utente'):

```
before_action :authenticate_user!
```

Per verificare se un utente ha effettuato l'accesso, utilizzare il seguente helper:

```
user_signed_in?
```

Per l'attuale utente che ha effettuato l'accesso, usa questo helper:

```
current_user
```

È possibile accedere alla sessione per questo ambito:

```
user_session
```

- Nota che se il tuo modello Devise è chiamato `Member` anziché `User` , sostituisci l' `user` sopra con il `member`

### Omniauth

Prima scegli la tua strategia di autenticazione e aggiungila al tuo `Gemfile` . Puoi trovare un elenco di strategie qui: https://github.com/intridea/omniauth/wiki/List-of-Strategies

```
gem 'omniauth-github', :github => 'intridea/omniauth-github'
gem 'omniauth-openid', :github => 'intridea/omniauth-openid'
```

Puoi aggiungerlo al middleware del tuo rails in questo modo:

```
Rails.application.config.middleware.use OmniAuth::Builder do
  require 'openid/store/filesystem'
  provider :github, ENV['GITHUB_KEY'], ENV['GITHUB_SECRET']
  provider :openid, :store => OpenID::Store::Filesystem.new('/tmp')
end
```

Per impostazione predefinita, OmniAuth aggiungerà `/auth/:provider` ai percorsi e puoi iniziare utilizzando questi percorsi.

Per impostazione predefinita, se si verifica un errore, omniauth reindirizzerà a `/auth/failure`

## has_secure_password

## Crea modello utente

```
rails generate model User email:string password_digest:string
```

## Aggiungi il modulo has_secure_password al modello Utente

```
class User < ActiveRecord::Base
  has_secure_password
end
```

Ora puoi creare un nuovo utente con password

```
user = User.new email: 'bob@bob.com', password: 'Password1', password_confirmation:
'Password1'
```

Verifica la password con il metodo di autenticazione

```
user.authenticate('somepassword')
```

## has_secure_token

Crea modello utente

```
# Schema: User(token:string, auth_token:string)
class User < ActiveRecord::Base
  has_secure_token
  has_secure_token :auth_token
end
```

Ora quando crei un nuovo utente un token e auth_token vengono generati automaticamente

```
user = User.new
user.save
user.token # => "pX27zsMN2ViQKta1bGfLmVJE"
user.auth_token # => "77TMHrHJFvFDwodq8w7Ev2m7"
```

Puoi aggiornare i token usando `regenerate_token` e `regenerate_auth_token`

```
user.regenerate_token # => true
user.regenerate_auth_token # => true
```

Leggi Autenticazione utente in Rails online: https://riptutorial.com/it/ruby-on-rails/topic/1794/autenticazione-utente-in-rails

# Capitolo 18: Autorizzazione con CanCan

## introduzione

CanCan è una semplice strategia di autorizzazione per Rails disaccoppiata dai ruoli utente. Tutte le autorizzazioni sono archiviate in un'unica posizione.

## Osservazioni

Prima di utilizzare CanCan, non dimenticare di creare utenti o inventare gem o manualmente. Per ottenere la massima funzionalità di CanCan, creare un utente amministratore.

## Examples

### Iniziare con CanCan

CanCan è una libreria di autorizzazione popolare per Ruby on Rails che limita l'accesso degli utenti a risorse specifiche. L'ultima gemma (CanCanCan) è la continuazione del progetto morto CanCan .

Le autorizzazioni sono definite nella classe `Ability` e possono essere utilizzate da controller, viste, helper o qualsiasi altra posizione nel codice.

Per aggiungere il supporto di autorizzazione a un'app, aggiungi la gemma CanCanCan al `Gemfile` :

```
gem 'cancancan'
```

Quindi definire la classe di abilità:

```
# app/models/ability.rb
class Ability
  include CanCan::Ability

  def initialize(user)
  end
end
```

Quindi verifica l'autorizzazione utilizzando `load_and_authorize_resource` per caricare i modelli autorizzati nel controller:

```
class ArticlesController < ApplicationController
  load_and_authorize_resource

  def show
    # @article is already loaded and authorized
  end
end
```

`authorize!` per verificare l'autorizzazione o sollevare un'eccezione

```
def show
  @article = Article.find(params[:id])
  authorize! :read, @article
end
```

`can?` per verificare se un oggetto è autorizzato contro una determinata azione in qualsiasi punto dei controller, delle viste o degli helper

```
<% if can? :update, @article %>
  <%= link_to "Edit", edit_article_path(@article) %>
<% end %>
```

**Nota: si** presume che l'utente firmato sia fornito dal metodo `current_user` .

## Definire le abilità

Le abilità sono definite nella classe `Ability` usando i metodi `can` e `cannot` . Considerare il seguente esempio commentato come riferimento di base:

```
class Ability
  include CanCan::Ability

  def initialize(user)
    # for any visitor or user
    can :read, Article

    if user
      if user.admin?
        # admins can do any action on any model or action
        can :manage, :all
      else
        # regular users can read all content
        can :read, :all
        # and edit, update and destroy their own user only
        can [:edit, :destroy], User, id: user_id
        # but cannot read hidden articles
        cannot :read, Article, hidden: true
      end
    else
      # only unlogged visitors can visit a sign_up page:
      can :read, :sign_up
    end
  end
end
```

## Gestire un gran numero di abilità

Una volta che il numero di definizioni di abilità inizia a crescere in numero, diventa sempre più difficile gestire il file Ability.

La prima strategia per gestire questi problemi è spostare le abilità in metodi significativi, come in questo esempio:

```
class Ability
  include CanCan::Ability

  def initialize(user)
    anyone_abilities

    if user
      if user.admin?
        admin_abilities
      else
        authenticated_abilities
      end
    else
      guest_abilities
    end
  end

  private

  def anyone_abilities
    # define abilities for everyone, both logged users and visitors
  end

  def guest_abilities
    # define abilities for visitors only
  end

  def authenticated_abilities
    # define abilities for logged users only
  end

  def admin_abilities
    # define abilities for admins only
  end
end
```

Una volta che questa classe è cresciuta abbastanza, puoi provare a suddividerla in classi diverse per gestire le diverse responsabilità come questa:

```
# app/models/ability.rb
class Ability
  include CanCan::Ability

  def initialize(user)
    self.merge Abilities::Everyone.new(user)

    if user
      if user.admin?
        self.merge Abilities::Admin.new(user)
      else
        self.merge Abilities::Authenticated.new(user)
      end
    else
      self.merge Abilities::Guest.new(user)
    end
  end
end
```

e quindi definire quelle classi come:

```
# app/models/abilities/guest.rb
module Abilities
  class Guest
    include CanCan::Ability

    def initialize(user)
      # Abilities for anonymous visitors only
    end
  end
end
```

e così via con `Abilities::Authenticated` , `Abilities::Admin` o qualsiasi altro.

### Prova rapidamente un'abilità

Se desideri testare rapidamente se una classe di abilità fornisce le autorizzazioni corrette, puoi inizializzare un'abilità nella console o in un altro contesto con l'ambiente rails caricato, basta passare un'istanza utente per testarla:

```
test_ability = Ability.new(User.first)
test_ability.can?(:show, Post) #=> true
other_ability = Ability.new(RestrictedUser.first)
other_ability.cannot?(:show, Post) #=> true
```

Ulteriori informazioni: https://github.com/ryanb/cancan/wiki/Testing-Abilities

Leggi Autorizzazione con CanCan online: https://riptutorial.com/it/ruby-on-rails/topic/3021/autorizzazione-con-cancan

# Capitolo 19: Blocco ActiveRecord

## Examples

### Blocco ottimistico

```
user_one = User.find(1)
user_two = User.find(1)

user_one.name = "John"
user_one.save
# Run at the same instance
user_two.name = "Doe"
user_two.save # Raises a ActiveRecord::StaleObjectError
```

### Blocco pessimistico

```
appointment = Appointment.find(5)
appointment.lock!
#no other users can read this appointment,
#they have to wait until the lock is released
appointment.save!
#lock is released, other users can read this account
```

Leggi Blocco ActiveRecord online: https://riptutorial.com/it/ruby-on-rails/topic/3866/blocco-activerecord

# Capitolo 20: caching

## Examples

### Caching di bambole russe

Si consiglia di nidificare i frammenti memorizzati nella cache all'interno di altri frammenti memorizzati nella cache. Questo è chiamato `Russian doll caching` .

Il vantaggio del `Russian doll caching` è che se un singolo prodotto viene aggiornato, tutti gli altri frammenti interni possono essere riutilizzati durante la rigenerazione del frammento esterno.

Come spiegato nella sezione precedente, un file memorizzato nella cache scadrà se il valore di `updated_at` cambia per un record dal quale dipende direttamente il file memorizzato nella cache. Tuttavia, questo non farà scadere alcuna cache all'interno della quale il frammento è annidato.

Ad esempio, prendere la seguente vista:

```
<% cache product do %>
  <%= render product.games %>
<% end %>
```

Che a sua volta rende questa vista:

```
<% cache game do %>
  <%= render game %>
<% end %>
```

Se viene modificato qualsiasi attributo di gioco, il valore `updated_at` verrà impostato sull'ora corrente, con conseguente scadenza della cache.

Tuttavia, poiché `updated_at` non verrà modificato per l'oggetto prodotto, tale cache non sarà scaduta e la tua app fornirà dati obsoleti. Per risolvere questo problema, leghiamo i modelli insieme al metodo touch:

```
class Product < ApplicationRecord
  has_many :games
end

class Game < ApplicationRecord
  belongs_to :product, touch: true
end
```

### Cache SQL

La cache di query è una funzionalità di `Rails` che memorizza nella cache il set di risultati restituito da ogni query. Se `Rails` riscontra nuovamente la stessa query per quella richiesta, utilizzerà il set di risultati memorizzato nella cache anziché eseguire nuovamente la query sul database.

Per esempio:

```
class ProductsController < ApplicationController

  def index
    # Run a find query
    @products = Product.all

    ...

    # Run the same query again
    @products = Product.all
  end

end
```

La seconda volta che la stessa query viene eseguita sul database, non sta andando a colpire il database. La prima volta che il risultato viene restituito dalla query, viene memorizzato nella cache della query (in memoria) e la seconda volta viene estratto dalla memoria.

Tuttavia, è importante notare che le cache di query vengono create all'inizio di un'azione e distrutte alla fine di tale azione e quindi rimangono attive solo per la durata dell'azione. Se desideri memorizzare i risultati della query in modo più persistente, puoi farlo con la memorizzazione nella cache di basso livello.

## Frammento di cache

`Rails.cache` , fornito da ActiveSupport, può essere utilizzato per memorizzare nella cache qualsiasi oggetto Ruby serializzabile tra le richieste.

Per recuperare un valore dalla cache per una determinata chiave, utilizzare `cache.read` :

```
Rails.cache.read('city')
# => nil
```

Usa `cache.write` per scrivere un valore nella cache:

```
Rails.cache.write('city', 'Duckburgh')
Rails.cache.read('city')
# => 'Duckburgh'
```

In alternativa, utilizza `cache.fetch` per leggere un valore dalla cache e facoltativamente scrivere un valore predefinito se non esiste alcun valore:

```
Rails.cache.fetch('user') do
  User.where(:is_awesome => true)
end
```

Il valore di ritorno del blocco passato verrà assegnato alla cache sotto la chiave specificata e quindi restituito.

Puoi anche specificare una scadenza cache:

```
Rails.cache.fetch('user', :expires_in => 30.minutes) do
  User.where(:is_awesome => true)
end
```

## Memorizzazione nella cache della pagina

Puoi utilizzare il pacchetto Action_Pack_caching gem per memorizzare nella cache le singole pagine. Questo memorizza il risultato di una richiesta dinamica come un file HTML statico, che viene servito al posto della richiesta dinamica sulle richieste successive. Il README contiene le istruzioni complete di impostazione. Una volta impostato, utilizzare il metodo di classe `caches_page` in un controller per memorizzare nella cache il risultato di un'azione:

```
class UsersController < ActionController::Base
  caches_page :index
end
```

Usa `expire_page` per forzare la scadenza della cache eliminando il file HTML memorizzato:

```
class UsersController < ActionController::Base
  caches_page :index

  def index
    @users = User.all
  end

  def create
    expire_page :action => :index
  end
end
```

La sintassi di `expire_page` riproduce quella di `url_for` e degli amici.

## Caching HTTP

Rails> = 3 viene fornito con le funzionalità di caching HTTP fuori dalla scatola. Questo utilizza le intestazioni `Cache-Control` e `ETag` per controllare per quanto tempo un client o un intermediario (come una CDN) può memorizzare nella cache una pagina.

In un'azione del controllore, usa `expires_in` per impostare la lunghezza della cache per quell'azione:

```
def show
  @user = User.find params[:id]
  expires_in 30.minutes, :public => true
end
```

Usa `expires_now` per forzare la scadenza immediata di una risorsa memorizzata nella cache su qualsiasi cliente o intermediario in visita:

```
def show
  @users = User.find params[:id]
  expires_now if params[:id] == 1
end
```

## Caching delle azioni

Come la memorizzazione nella cache delle pagine, il caching delle azioni memorizza nella cache l'intera pagina. La differenza è che la richiesta colpisce lo stack Rails, quindi prima che i filtri vengano eseguiti prima che la cache venga pubblicata. Viene estratto da Rails nella gemma actionpack-action_caching .

Un esempio comune è il caching di un'azione che richiede l'autenticazione:

```
class SecretIngredientsController < ApplicationController
  before_action :authenticate_user!, only: :index, :show
  caches_action :index

  def index
    @secret_ingredients = Recipe.find(params[:recipe_id]).secret_ingredients
  end
end
```

Le opzioni includono `:expires_in` , una custom `:cache_path` (per le azioni con più route che devono essere memorizzate nella cache in modo diverso) e `:if` / `:unless` che `:unless` controlli quando l'azione deve essere memorizzata nella cache.

```
class RecipesController < ApplicationController
  before_action :authenticate_user!, except: :show
  caches_page :show
  caches_action :archive, expires_in: 1.day
  caches_action :index, unless: { request.format.json? }
end
```

Quando il layout ha un contenuto dinamico, memorizza nella cache solo il contenuto dell'azione passando il `layout: false` .

Leggi caching online: https://riptutorial.com/it/ruby-on-rails/topic/2833/caching

# Capitolo 21: Colonne ActiveRecord multiuso

## Sintassi

- `serialize: <field_plural_symbol>`

## Examples

### Salvataggio di un oggetto

Se si dispone di un attributo che deve essere salvato e recuperato nel database come oggetto, specificare il nome di tale attributo utilizzando il metodo `serialize` e verrà gestito automaticamente.

L'attributo deve essere dichiarato come campo di `text` .

Nel modello devi dichiarare il tipo di campo ( `Hash` o `Array` )

Maggiori informazioni su: serialize >> apidock.com

### Come

# Nella tua migrazione

```
class Users < ActiveRecord::Migration[5.0]
  def change
    create_table :users do |t|
      ...
      t.text :preference
      t.text :tag
      ...
      t.timestamps
    end
  end
end
```

# Nel tuo modello

```
class User < ActiveRecord::Base
    serialize :preferences, Hash
    serialize :tags, Array
end
```

Leggi Colonne ActiveRecord multiuso online: https://riptutorial.com/it/ruby-on-rails/topic/7602/colonne-activerecord-multiuso

# Capitolo 22: Configura Angolare con Rails

## Examples

### Angolare con Rails 101

# Passaggio 1: crea una nuova app di Rails

```
gem install rails -v 4.1
rails new angular_example
```

# Passaggio 2: rimuovere i turbolinks

Per rimuovere i turbolinks è necessario rimuoverlo dal Gemfile.

```
gem 'turbolinks'
```

Rimuovi la `require` da `app/assets/javascripts/application.js` :

```
//= require turbolinks
```

# Passaggio 3: aggiungere AngularJS alla pipeline degli asset

Per far funzionare Angular con la pipeline di asset di Rails, dobbiamo aggiungere al Gemfile:

```
gem 'angular-rails-templates'
gem 'bower-rails'
```

Ora esegui il comando

```
bundle install
```

Aggiungi `bower` modo da poter installare la dipendenza AngularJS:

```
rails g bower_rails:initialize json
```

Aggiungi Angolare a `bower.json` :

```
{
```

```
  "name": "bower-rails generated dependencies",

  "dependencies": {

    "angular": "latest",
    "angular-resource": "latest",
    "bourbon": "latest",
    "angular-bootstrap": "latest",
    "angular-ui-router": "latest"
  }
}
```

Ora che `bower.json` è configurato con le giuste dipendenze, installiamolo:

```
bundle exec rake bower:install
```

# Passaggio 4: organizzazione dell'app Angular

Crea la seguente struttura di cartelle in `app/assets/javascript/angular-app/` :

```
templates/
modules/
filters/
directives/
models/
services/
controllers/
```

In `app/assets/javascripts/application.js` , aggiungi `require` per Angular, l'helper del template e la struttura del file dell'app Angular. Come questo:

```
//= require jquery
//= require jquery_ujs

//= require angular
//= require angular-rails-templates
//= require angular-app/app

//= require_tree ./angular-app/templates
//= require_tree ./angular-app/modules
//= require_tree ./angular-app/filters
//= require_tree ./angular-app/directives
//= require_tree ./angular-app/models
//= require_tree ./angular-app/services
//= require_tree ./angular-app/controllers
```

# Passaggio 5: avviare l'app Angolare

Crea `app/assets/javascripts/angular-app/app.js.coffee` :

```
@app = angular.module('app', [ 'templates' ])
```

```
@app.config([ '$httpProvider', ($httpProvider)->
$httpProvider.defaults.headers.common['X-CSRF-Token'] =
$('meta[name=csrftoken]').attr('content') ])  @app.run(->   console.log 'angular app running'
)
```

Creare un modulo angolare su `app/assets/javascripts/angular-app/modules/example.js.coffee.erb` :

```
@exampleApp = angular.module('app.exampleApp', [     # additional dependencies here   ])
.run(->     console.log 'exampleApp running'   )
```

Crea un controller angolare per questa app su `app/assets/javascripts/angular-app/controllers/exampleCtrl.js.coffee` :

```
angular.module('app.exampleApp').controller("ExampleCtrl", [   '$scope',   ($scope)->
console.log 'ExampleCtrl running'      $scope.exampleValue = "Hello angular and rails"  ])
```

Ora aggiungi un percorso a Rails per passare il controllo ad Angular. In `config/routes.rb` :

```
Rails.application.routes.draw do   get 'example' => 'example#index' end
```

Genera il controller Rails per rispondere a questa rotta:

```
rails g controller Example
```

In `app/controllers/example_controller.rb` :

```
class ExampleController < ApplicationController
    def index
    end
end
```

Nella vista, dobbiamo specificare quale app angolare e quale controller angolare guiderà questa pagina. Quindi in `app/views/example/index.html.erb` :

```
<div ng-app='app.exampleApp' ng-controller='ExampleCtrl'>

  <p>Value from ExampleCtrl:</p>
  <p>{{ exampleValue }}</p>

</div>
```

Per visualizzare l'app, avviare il server Rails e visitare http: // localhost: 3000 / example .

Leggi Configura Angolare con Rails online: https://riptutorial.com/it/ruby-on-rails/topic/3902/configura-angolare-con-rails

# Capitolo 23: Configurazione

## Examples

### Configurazione personalizzata

Creare un file `YAML` nella directory `config/` , ad esempio: `config/neo4j.yml`

Il contenuto di `neo4j.yml` può essere qualcosa di simile al seguente (per semplicità, l' `default` è utilizzata per tutti gli ambienti):

```
default: &default
  host: localhost
  port: 7474
  username: neo4j
  password: root

development:
  <<: *default

test:
  <<: *default

production:
  <<: *default
```

in `config/application.rb` :

```
module MyApp
  class Application < Rails::Application
    config.neo4j = config_for(:neo4j)
  end
end
```

Ora, la tua configurazione personalizzata è accessibile come di seguito:

```
Rails.configuration.neo4j['host']
 #=> localhost
Rails.configuration.neo4j['port']
 #=> 7474
```

---

#### Ulteriori informazioni

Il documento API ufficiale di Rails descrive il metodo `config_for` come:

> Convenienza per caricare config / foo.yml per l'attuale Rails env.

---

#### Se non vuoi usare un file `yaml`

È possibile configurare il proprio codice tramite l'oggetto di configurazione di Rails con configurazione personalizzata nella proprietà `config.x` .

**Esempio**

```
config.x.payment_processing.schedule = :daily
config.x.payment_processing.retries  = 3
config.x.super_debugger = true
```

Questi punti di configurazione sono quindi disponibili tramite l'oggetto di configurazione:

```
Rails.configuration.x.payment_processing.schedule # => :daily
Rails.configuration.x.payment_processing.retries  # => 3
Rails.configuration.x.super_debugger              # => true
Rails.configuration.x.super_debugger.not_set      # => nil
```

Leggi Configurazione online: https://riptutorial.com/it/ruby-on-rails/topic/2558/configurazione

# Capitolo 24: Configurazione

## Examples

### Ambienti in Rails

I file di configurazione per i binari possono essere trovati in `config/environments/` . Di default le rotaie hanno 3 ambienti, `development` , `production` e `test` . Modificando ogni file si modifica la configurazione solo per quell'ambiente.

Rails ha anche un file di `config/application.rb` in `config/application.rb` . Questo è un file di configurazione comune poiché le impostazioni qui definite vengono sovrascritte dalla configurazione specificata in ogni ambiente.

Aggiungete o modificate le opzioni di configurazione all'interno di `Rails.application.configure do` blocco e le opzioni di configurazione iniziano con `config`.

### Configurazione del database

La configurazione del database di un progetto di rotaie si trova in un file `config/database.yml` . Se si crea un progetto utilizzando il comando `rails new` e non si specifica un motore di database da utilizzare, rails utilizza `sqlite` come database predefinito. Un tipico file `database.yml` con configurazione predefinita sarà simile al seguente.

```
# SQLite version 3.x
#   gem install sqlite3
#
#   Ensure the SQLite 3 gem is defined in your Gemfile
#   gem 'sqlite3'
#
default: &default
  adapter: sqlite3
  pool: 5
  timeout: 5000

development:
  <<: *default
  database: db/development.sqlite3

# Warning: The database defined as "test" will be erased and
# re-generated from your development database when you run "rake".
# Do not set this db to the same as development or production.
test:
  <<: *default
  database: db/test.sqlite3

production:
  <<: *default
  database: db/production.sqlite3
```

Se vuoi cambiare il database predefinito mentre crei un nuovo progetto puoi specificare il

database: `rails new hello_world --database=mysql`

## Configurazione generale delle guide

Le seguenti opzioni di configurazione dovrebbero essere chiamate su un oggetto `Rails::Railtie`

- **config.after_initialize** : prende un blocco che verrà eseguito dopo che le rotaie hanno inizializzato l'applicazione.
- **config.asset_host** : imposta l'host per le risorse. Questo è utile quando si utilizza una *rete di consegna del contenuto* . Questa è una scorciatoia per `config.action_controller.asset_host`
- **config.autoload_once_paths** : questa opzione accetta una serie di percorsi in cui Rails **esegue il** caricamento automatico delle costanti. Il valore predefinito è un array vuoto
- **config.autoload_paths** : accetta una serie di percorsi in cui Rails **esegue il** caricamento automatico delle costanti. Per impostazione predefinita tutte le directory in `app`
- **config.cache_classes** : determina se le classi e i moduli devono essere ricaricati su ogni richiesta. Nella modalità di sviluppo, questa impostazione predefinita è `false` e nelle modalità di produzione e di test è impostata su `true`
- **config.action_view.cache_template_loading** : determina se i modelli devono essere ricaricati su ogni richiesta. Il valore predefinito è l'impostazione `config.cache_classes`
- **config.beginning_of_week** : imposta l'inizio predefinito della settimana. Richiede un simbolo valido per il giorno della settimana ( `:monday` )
- **config.cache_store** : scegli quale negozio di cache usare. Le opzioni includono `:file_store` `:memory_store` , `mem_cache_store` o `null_store` .
- **config.colorize_logging** : controlla se le informazioni di registrazione sono colorate
- **config.eager_load** : Eager-carica tutto registrato
- **config.encoding** : specifica la codifica dell'applicazione. Il valore predefinito è `UTF-8`
- **config.log_level** : imposta la verbosità di Rails Logger. Per impostazione predefinita, `:debug` in tutti gli ambienti.
- **config.middleware** : utilizzare questo per configurare il middleware dell'applicazione
- **config.time_zone** : imposta il fuso orario predefinito dell'applicazione.

## Configurazione delle risorse

Le seguenti opzioni di configurazione possono essere utilizzate per la configurazione delle risorse

- **config.assets.enabled** : determina se la pipeline degli asset è abilitata. Questo valore predefinito è true
- **config.assets.raise_runtime_errors** : abilita il controllo degli errori di runtime. È utile per la `development mode`
- **config.assets.compress** : consente di comprimere le risorse. Nella modalità di produzione, questa impostazione predefinita è true
- **config.assets.js_compressor** : specifica quale compressore JS usare. Le opzioni includono `:closure` `:uglifier` e `:yui`
- **config.assets.paths** : specifica quali percorsi cercare i beni.
- **config.assets.precompile** : consente di selezionare asset aggiuntivi da precompilare durante il `rake assets:precompile` viene eseguito il `rake assets:precompile`
- **config.assets.digest** : questa opzione consente l'uso di impronte digitali `MD-5` nei nomi degli

asset. Il valore predefinito è true in modalità sviluppo
- **config.assets.compile** : Attiva la compilazione di `Sprockets` dal vivo in modalità produzione

## Configurazione dei generatori

Rails consente di configurare quali generatori vengono utilizzati durante l'esecuzione di comandi di `rails generate` . Questo metodo, `config.generators` prende un blocco

```
config.generators do |g|
  g.orm :active_record
  g.test_framework :test_unit
end
```

Ecco alcune delle opzioni

| Opzione | Descrizione | Predefinito |
|---|---|---|
| risorse | Crea risorse durante la generazione di scaffold | vero |
| force_plural | Consente nomi di modelli pluralizzati | falso |
| aiutante | Determina se generare helper | vero |
| integration_tool | Specificare lo strumento di integrazione | `test_unit` |
| javascript_engine | Configura il motore JS | `:js` |
| resource_route | Genera rotta delle risorse | vero |
| stylesheet_engine | Configura il motore del foglio di stile | `:cs` |
| scaffold_stylesheet | Crea CSS su scaffolding | vero |
| test_framework | Specifica il framework di test | `Minitest` |
| template_engine | Configura il motore di template | `:erb` |

Leggi Configurazione online: https://riptutorial.com/it/ruby-on-rails/topic/2841/configurazione

# Capitolo 25: Constantize costante

## Examples

### Sicuro_constantize riuscito

`User` è una classe `ActiveRecord` o `Mongoid` . Sostituisci `User` con qualsiasi classe `Rails` nel tuo progetto (anche qualcosa come `Integer` o `Array` )

```
my_string = "User" # Capitalized string
  # => 'User'
my_constant = my_string.safe_constantize
  # => User
my_constant.all.count
  # => 18

my_string = "Array"
  # => 'Array'
my_constant = my_string.safe_constantize
  # => Array
my_constant.new(4)
  # => [nil, nil, nil, nil]
```

### Safe_constantize non riuscito

Questo esempio non funzionerà perché la stringa passata non è riconosciuta come costante nel progetto. Anche se si passa in `"array"` , non funzionerà in quanto non è in maiuscolo.

```
my_string = "not_a_constant"
  # => 'not_a_constant'
my_string.safe_constantize
  # => nil

my_string = "array" #Not capitalized!
  # => 'array'
my_string.safe_constantize
  # => nil
```

Leggi Constantize costante online: https://riptutorial.com/it/ruby-on-rails/topic/3015/constantize-costante

# Capitolo 26: Convenzioni di denominazione

## Examples

### Controller

I nomi delle classi controller sono pluralizzati. Il motivo è che il controllore controlla più istanze di istanza di oggetto.

*Ad esempio* : `OrdersController` sarebbe il controller per una tabella di `orders` . Rails cercherà quindi la definizione della classe in un file chiamato `orders_controller.rb` nella `orders_controller.rb` `/app/controllers` .

*Ad esempio* : `PostsController` sarebbe il controller per una tabella di `posts` .

Se il nome della classe controller ha più parole in maiuscolo, si presume che il nome della tabella abbia underscore tra queste parole.

*Ad esempio:* se un controller è denominato `PendingOrdersController` il nome file presunto per questo controller sarà `pending_orders_controller.rb` .

### Modelli

Il modello è denominato utilizzando la convenzione di denominazione della classe di MixedCase non interrotto ed è sempre il singolare del nome della tabella.

*Ad esempio* : se una tabella è stata denominata `orders` , il modello associato sarà denominato `Order`

*Ad esempio* : se una tabella è stata denominata `posts` , il modello associato sarà denominato `Post`

Rails cercherà quindi la definizione della classe in un file chiamato `order.rb` nella `order.rb` `/app/models` .

Se il nome della classe del modello ha più parole in maiuscolo, si presume che il nome della tabella abbia underscore tra queste parole.

*Ad esempio:* se un modello è denominato `BlogPost` , il nome di tabella presunto sarà `blog_posts` .

### Viste e layout

Quando viene eseguita un'azione di controller, Rails tenterà di trovare un layout e una visualizzazione corrispondenti in base al nome del controller.

Le viste e i layout sono posizionati nella directory `app/views` .

Data una richiesta all'azione `PeopleController#index` , Rails cercherà:

- il layout chiamato `people` in `app/views/layouts/` (o `application` se non viene trovata alcuna corrispondenza)
- una vista chiamata `index.html.erb` in `app/views/people/` per impostazione predefinita
- se desideri eseguire il rendering di un altro file chiamato `index_new.html.erb` devi scrivere il codice per questo nell'azione `PeopleController#index render 'index_new'` `PeopleController#index` come `render 'index_new'`
- possiamo impostare diversi `layouts` per ogni `action` scrivendo `render 'index_new', layout: 'your_layout_name'`

## Nomi di file e autoloading

I file Rails - e i file Ruby in generale - dovrebbero essere nominati con `lower_snake_case` file `lower_snake_case` . Per esempio

```
app/controllers/application_controller.rb
```

è il file che contiene la definizione della classe `ApplicationController` . Si noti che mentre `PascalCase` viene utilizzato per i nomi di classi e moduli, i file in cui risiedono dovrebbero essere ancora `lower_snake_case` .

La denominazione coerente è importante poiché Rails utilizza i file di caricamento automatico secondo necessità e utilizza "flection" per trasformare tra diversi stili di denominazione, come la trasformazione di `application_controller` in `ApplicationController` e viceversa.

Ad esempio, se Rails `BlogPost` che la classe `BlogPost` non esiste (non è stata ancora caricata), cercherà un file denominato `blog_post.rb` e tenterà di caricare quel file.

È quindi anche importante assegnare un nome ai file per ciò che contengono, poiché il caricatore automatico si aspetta che i nomi dei file corrispondano al contenuto. Se, ad esempio, `blog_post.rb` contiene invece una classe chiamata solo `Post` , verrà visualizzato un `LoadError` : `Expected [some path]/blog_post.rb to define BlogPost` .

Se aggiungi una dir sotto `app/something/` (ad esempio / models / products /), e

- vuoi creare un namespace per i moduli e le classi all'interno della nuova cartella, quindi non devi fare nulla e verrà caricato da solo. Ad esempio, in `app/models/products/ you would need to wrap your class in` modulo Prodotti`.
- non voglio creare un namespace per i moduli e le classi all'interno della mia nuova `config.autoload_paths += %W( #{config.root}/app/models/products )` quindi devi aggiungere `config.autoload_paths += %W( #{config.root}/app/models/products )` al tuo `application.rb` per autoload.

Un'altra cosa a cui prestare attenzione (specialmente se l'inglese non è la tua prima lingua) è il fatto che Rails conti nomi irregolari plurali in inglese. Quindi se hai un modello chiamato "Piede", il controller corrispondente deve essere chiamato "FeetController" piuttosto che "FootsController" se vuoi che il routing "magico" delle rotte (e molte altre di tali caratteristiche) funzioni.

## Classe di modelli dal nome del controller

È possibile ottenere una classe Model da un nome Controller in questo modo (il contesto è classe Controller):

```
class MyModelController < ActionController::Base

  # Returns corresponding model class for this controller
  # @return [ActiveRecord::Base]
  def corresponding_model_class
    # ... add some validation
    controller_name.classify.constantize
  end
end
```

Leggi Convenzioni di denominazione online: https://riptutorial.com/it/ruby-on-rails/topic/1493/convenzioni-di-denominazione

# Capitolo 27: Debug

## Examples

### Debugging Rails Application

Per poter eseguire il debug di un'applicazione è molto importante comprendere il flusso della logica e dei dati di un'applicazione. Aiuta a risolvere bug logici e aggiunge valore all'esperienza di programmazione e alla qualità del codice. Due gemme popolari per il debugging sono debugger (per ruby 1.9.2 e 1.9.3) e byebug (per ruby> = 2.x).

Per il debug di file `.rb` , attenersi alla seguente procedura:

1. Aggiungi `debugger` o `byebug` al gruppo di `development` di `Gemfile`
2. Esegui `bundle install`
3. Aggiungi `debugger` o `byebug` come breakpoint
4. Esegui il codice o fai richiesta
5. Vedere il registro del server delle guide interrotto nel punto di interruzione specificato
6. A questo punto puoi utilizzare il tuo terminale server proprio come la `rails console` e controllare i valori di variabile e params
7. Per passare all'istruzione successiva, digitare `next` e premere `enter`
8. Per uscire da digitare `c` e premere `enter`

Se si desidera eseguire il debug `.html.erb` file `.html.erb` , il punto di interruzione verrà aggiunto come `<% debugger %>`

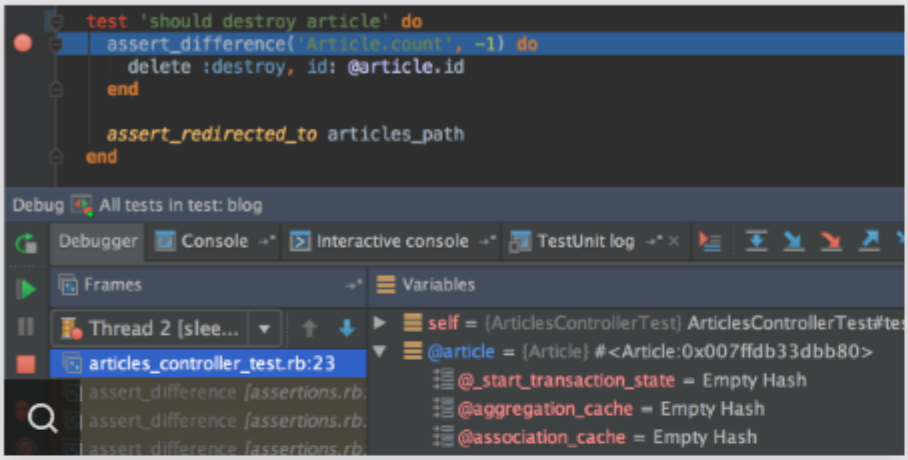
### Debugging nel tuo IDE

Ogni buon IDE fornisce una GUI per il debugging interattivo di applicazioni Ruby (e quindi Rails) in cui è possibile aggiungere punti di interruzione, orologi, pausa automatica su eccezioni e consente di seguire l'esecuzione del codice anche passo dopo passo, riga per riga.

Ad esempio, dai uno sguardo alle migliori funzioni di debug di Ruby IDE e RubyMine sull'immagine

# Powerful Debugger

RubyMine brings a clever debugger with a graphical UI for Ruby, JS, and CoffeeScript. Set breakpoints and run your code step by step with all the information at your fingertips.




## Convenient user in

- Look under the hood of ar Variables and Watches vie
- The UI is fully customizabl select toolbar commands, project code while steppir
- The debugger UI is also tig navigate between the deb etc.
- You also get the complete views.

## Smart, flexible breakpoints

- Place a breakpoint on a line of code and define the hit conditions—a set of Boolean expressions that are evaluated to determine whether to stop the code execution or not.
- If you have multiple breakpoints in your code, you can set dependencies between them to define the order in which they can be hit.
- Setting a breakpoint is just a matter of a single mouse click on the gutter or invoking a shortcut.
- Breakpoints are also available in Rails views, so you can use them for debugging Rails code as well.

## Debugging JavaSc

- RubyMine provides an adv which works with Google
- You can easily debug ECM on RubyMine debugger's
- A full-featured debugger to debug apps running loc

## Built-in expression evaluator

Evaluate any expression while your debugging session is paused. Type an expression or a code fragment, with coding assistance available in the dialog. All expressions are evaluated against the current context.

## Dedicated Watches

Track any number of expressio current stack frame context. T through your debugging sessi

## Frames and call stack

When a breakpoint is hit or code execution is suspended, you can use the Frames panel to examine the current threads, their state, call stack, methods, and variables along with their values.

## Remote debugging

As you connect to a remote ho mapping between the local so debug processes can be launc

***Il debugging sollevando eccezioni*** è *molto più semplice* dello strizzare gli occhi attraverso `print` istruzioni del registro di `print` e, per la maggior parte degli errori, è generalmente *molto più veloce* dell'apertura di un debugger irb come `pry` o `byebug` . Questi strumenti non dovrebbero essere il tuo primo passo.

---

# Eseguire il debug di Ruby / Rails rapidamente:

## 1. Metodo veloce: Alza un `Exception` allora e `.inspect` suo risultato

Il modo *più veloce* per eseguire il debug del codice Ruby (in particolare Rails) consiste nel `raise` un'eccezione lungo il percorso di esecuzione del codice durante la chiamata a `.inspect` sul metodo o sull'oggetto (ad es. `foo` ):

```
raise foo.inspect
```

Nel codice precedente, *`raise`* innesca *`Exception`* che *interrompe l'esecuzione del codice* e restituisce un messaggio di errore che contiene convenientemente informazioni `.inspect` sull'oggetto / metodo (cioè `foo` ) sulla riga che si sta tentando di eseguire il debug.

Questa tecnica è utile per esaminare *rapidamente* un oggetto o un metodo ( *ad esempio è `nil` ?* ) E per confermare immediatamente se una linea di codice viene addirittura eseguita in un dato contesto.

## 2. Fallback: usa un debugger *IRB* rubino come `byebug` o `pry`

Solo dopo aver ricevuto informazioni sullo stato del flusso di esecuzione dei codici, dovresti prendere in considerazione la possibilità di passare a un debugger di ruby gem irb come `pry` o `byebug` cui è possibile approfondire lo stato degli oggetti all'interno del percorso di esecuzione.

Per utilizzare la gem di `byebug` per il debug in Rails:

1. Aggiungi `gem 'byebug'` all'interno del gruppo di *sviluppo* nel tuo *Gemfile*
2. Esegui `bundle install`
3. Quindi per utilizzare, inserire la frase `byebug` all'interno del percorso di esecuzione del codice che si desidera esaminare.

Questa variabile di `byebug` quando eseguita aprirà una sessione IRB ruby del tuo codice, dandoti l'accesso diretto allo stato degli oggetti così come sono a quel punto nell'esecuzione del codice.

I debugger IRB come Byebug sono utili per analizzare in profondità lo stato del tuo codice durante l'esecuzione. Tuttavia, sono procedure che richiedono più tempo rispetto all'aumento degli errori,

quindi nella maggior parte delle situazioni non dovrebbero essere il primo passo.

---

# Consigli generali per principianti

Quando si tenta di eseguire il debug di un problema, un buon consiglio è sempre: **leggere il messaggio di errore! @ # $ Errore (RTFM)**

Ciò significa che la lettura dei messaggi di errore *con attenzione* e *completamente* prima di agire in modo da *capire che cosa sta cercando di dirti.* Quando esegui il debug, poni le seguenti domande mentali, *in questo ordine* , durante la lettura di un messaggio di errore:

1. Quale ***classe*** fa riferimento all'errore? (cioè *ho la classe dell'oggetto corretta o il mio oggetto è `nil` ?* )
2. Quale ***metodo*** fa riferimento l'errore? (cioè *è il loro un tipo nel metodo, posso chiamare questo metodo su questo tipo / classe di oggetto?* )
3. Infine, usando ciò che posso dedurre dalle mie ultime due domande, quali ***linee di codice*** dovrei investigare? (ricorda: l'ultima riga di codice nella traccia dello stack non è necessariamente dove si trova il problema).

Nella traccia dello stack prestate particolare attenzione alle righe di codice che provengono dal vostro progetto (es. Le righe che iniziano con `app/...` se state usando Rails). Il 99% delle volte il problema è con il tuo codice.

---

Per illustrare perché interpretare *in questo ordine* è importante ...

### Ad esempio un messaggio di errore Ruby che confonde molti principianti:

Esegui codice che ad un certo punto viene eseguito come tale:

```
@foo = Foo.new

...

@foo.bar
```

e ottieni un errore che afferma:

```
undefined method "bar" for Nil:nilClass
```

I principianti vedono questo errore e pensano che il problema sia che la `bar` del metodo *non* è *definita* . **Non è.** In questo errore la parte reale che conta è:

```
for Nil:nilClass
```

**`for Nil:nilClass` significa che `@foo` è Nil!** `@foo` non è una variabile di istanza di `Foo` ! Hai un oggetto che è `Nil` . Quando vedi questo errore, è semplicemente rubino cercando di dirti che la `bar` del metodo non esiste per oggetti della classe `Nil` . (beh, perché stiamo cercando di usare un

metodo per un oggetto della classe `Foo` not `Nil` ).

Sfortunatamente, a causa di come viene scritto questo errore ( `undefined method "bar" for Nil:nilClass` ) è facile essere indotti a pensare che questo errore abbia a che fare con la `bar undefined` . Quando non viene letto attentamente, questo errore induce i principianti a scavare erroneamente nei dettagli del metodo `bar` su `Foo` , mancando completamente la parte dell'errore che suggerisce che l'oggetto è della classe sbagliata (in questo caso: nil). È un errore che è facilmente evitato leggendo i messaggi di errore nella loro interezza.

**Sommario:**

**Leggere** sempre attentamente **l' *intero* messaggio di errore** prima di iniziare qualsiasi debug. Ciò significa: Controllare sempre il tipo di ***classe*** di un oggetto in un messaggio di *errore,* poi i suoi ***metodi,*** *prima* di iniziare sleuthing in qualsiasi stacktrace o riga di codice in cui si pensa l'errore può essere che si verificano. Quei 5 secondi possono farti risparmiare 5 ore di frustrazione.

**tl; dr:** Non strizzare gli occhi ai registri di stampa: alza invece le eccezioni. Evita buchi di coniglio leggendo gli errori attentamente prima di eseguire il debug.

## Eseguire il debug dell'applicazione ruby-on-rails con leva

leva è uno strumento potente che può essere utilizzato per eseguire il debug di qualsiasi applicazione di rubino. Impostare un'applicazione ruby-on-rails con questa gemma è molto semplice e diretto.

**Impostare**

Per avviare il debug della tua applicazione con leva

- Aggiungi `gem 'pry'` al `Gemfile` dell'applicazione e raggruppalo

```
group :development, :test do
  gem 'pry'
end
```

- Passare alla directory principale dell'applicazione sulla console del terminale ed eseguire l' `bundle install` . Sei pronto per iniziare a utilizzarlo ovunque nella tua applicazione.

**Uso**

L'uso di leva nella tua applicazione include solo `binding.pry` sui punti di interruzione che vuoi ispezionare durante il debug. È possibile aggiungere i breakpoint di `binding.pry` qualsiasi punto dell'applicazione che viene interpretato dall'interprete ruby (qualsiasi app / controller, app / modelli, file app / view)

i) Debug di un controller

*app / controllers / users_controller.rb*

```
class UsersController < ApplicationController
```

```
  def show
    use_id = params[:id]
    // breakpoint to inspect if the action is receiving param as expected
    binding.pry
    @user = User.find(user_id)
    respond_to do |format|
      format.html
    end
  end
end
```

In questo esempio, il server delle guide si interrompe con una console di leva al punto di interruzione quando si tenta di visitare un routing della pagina per `show` azione su `UsersController` . È possibile esaminare l'oggetto `params` e rendere la query ActiveRecord sul modello `User` da quel punto di interruzione

ii) Debug di una vista

*app / views / utenti / show.html.haml*

```
%table
  %tbody
    %tr
      %td ID
      %td= @user.id
    %tr
      %td email
      %td= @user.email
    %tr
      %td logged in ?
      %td
        - binding.pry
        - if @user.logged_in?
          %p= "Logged in"
        - else
          %p= "Logged out"
```

In questo esempio, il punto di interruzione si interrompe con la console di leva quando la pagina degli `users/show` è precompilata nel server di rotaie prima di inviarla al browser del client. Questo break-point consente di correggere la correttezza di `@user.logged_in?` quando si comporta male.

ii) Debug di un modello

```
app/models/user.rb

class User < ActiveRecord::Base
  def full_name
    binding.pry
    "#{self.first_name} #{self.last_name}"
  end
end
```

In questo esempio, il punto di interruzione può essere utilizzato per eseguire il debug del metodo di istanza del modello `User` `full_name` quando questo metodo viene chiamato da qualsiasi punto dell'applicazione.

In conclusione, la leva è un potente strumento di debug per l'applicazione rails con una semplice impostazione e linee guida di debug semplici. Fai un tentativo.

Leggi Debug online: https://riptutorial.com/it/ruby-on-rails/topic/3877/debug

# Capitolo 28: Distribuzione di un'app Rails su Heroku

## Examples

### Distribuzione della tua applicazione

Assicurati di essere nella directory che contiene l'app Rails, quindi crea un'app su Heroku.

```
$ heroku create example
Creating  example... done
https://example.herokuapp.com/ | https://git.heroku.com/example.git
```

Il primo URL dell'output, http://example.herokuapp.com , è la posizione in cui è disponibile l'app. Il secondo URL, git@heroku.com: example.git, è l'URL del repository git remoto.

Questo comando dovrebbe essere utilizzato solo su un repository git inizializzato. Il comando heroku create aggiunge automaticamente un git remote chiamato "heroku" che punta a questo URL.

L'argomento del nome dell'app ("esempio") è facoltativo. Se non viene specificato alcun nome app, verrà generato un nome casuale. Poiché i nomi delle app di Heroku si trovano in uno spazio dei nomi globale, è possibile che vengano già utilizzati nomi comuni, come "blog" o "wiki". È spesso più facile iniziare con un nome predefinito e rinominare l'app in un secondo momento.

Quindi, distribuisci il tuo codice:

```
$ git push heroku master
remote: Compressing source files... done.
remote: Building source:
remote:
remote: -----> Ruby app detected
remote: -----> Compiling Ruby/Rails
remote: -----> Using Ruby version: ruby-2.3.1
remote: -----> Installing dependencies using bundler 1.11.2
remote:        Running: bundle install --without development:test --path vendor/bundle --
binstubs vendor/bundle/bin -j4 --deployment
remote:        Warning: the running version of Bundler is older than the version that created
the lockfile. We suggest you upgrade to the latest version of Bundler by running `gem install
bundler`.
remote:        Fetching gem metadata from https://rubygems.org/..........
remote:        Fetching version metadata from https://rubygems.org/...
remote:        Fetching dependency metadata from https://rubygems.org/..
remote:        Installing concurrent-ruby 1.0.2
remote:        Installing i18n 0.7.0
remote:        Installing rake 11.2.2
remote:        Installing minitest 5.9.0
remote:        Installing thread_safe 0.3.5
remote:        Installing builder 3.2.2
remote:        Installing mini_portile2 2.1.0
```

```
remote:        Installing erubis 2.7.0
remote:        Installing pkg-config 1.1.7
remote:        Installing rack 2.0.1
remote:        Installing nio4r 1.2.1 with native extensions
remote:        Installing websocket-extensions 0.1.2
remote:        Installing mime-types-data 3.2016.0521
remote:        Installing arel 7.0.0
remote:        Installing coffee-script-source 1.10.0
remote:        Installing execjs 2.7.0
remote:        Installing method_source 0.8.2
remote:        Installing thor 0.19.1
remote:        Installing multi_json 1.12.1
remote:        Installing puma 3.4.0 with native extensions
remote:        Installing pg 0.18.4 with native extensions
remote:        Using bundler 1.11.2
remote:        Installing sass 3.4.22
remote:        Installing tilt 2.0.5
remote:        Installing turbolinks-source 5.0.0
remote:        Installing tzinfo 1.2.2
remote:        Installing nokogiri 1.6.8 with native extensions
remote:        Installing rack-test 0.6.3
remote:        Installing sprockets 3.6.3
remote:        Installing websocket-driver 0.6.4 with native extensions
remote:        Installing mime-types 3.1
remote:        Installing coffee-script 2.4.1
remote:        Installing uglifier 3.0.0
remote:        Installing turbolinks 5.0.0
remote:        Installing activesupport 5.0.0
remote:        Installing mail 2.6.4
remote:        Installing globalid 0.3.6
remote:        Installing activemodel 5.0.0
remote:        Installing jbuilder 2.5.0
remote:        Installing activejob 5.0.0
remote:        Installing activerecord 5.0.0
remote:        Installing loofah 2.0.3
remote:        Installing rails-dom-testing 2.0.1
remote:        Installing rails-html-sanitizer 1.0.3
remote:        Installing actionview 5.0.0
remote:        Installing actionpack 5.0.0
remote:        Installing actionmailer 5.0.0
remote:        Installing railties 5.0.0
remote:        Installing actioncable 5.0.0
remote:        Installing sprockets-rails 3.1.1
remote:        Installing coffee-rails 4.2.1
remote:        Installing jquery-rails 4.1.1
remote:        Installing rails 5.0.0
remote:        Installing sass-rails 5.0.5
remote:        Bundle complete! 15 Gemfile dependencies, 54 gems now installed.
remote:        Gems in the groups development and test were not installed.
remote:        Bundled gems are installed into ./vendor/bundle.
remote:        Bundle completed (31.86s)
remote:        Cleaning up the bundler cache.
remote:        Warning: the running version of Bundler is older than the version that created
the lockfile. We suggest you upgrade to the latest version of Bundler by running `gem install
bundler`.
remote: -----> Preparing app for Rails asset pipeline
remote:        Running: rake assets:precompile
remote:        I, [2016-07-08T17:08:57.046245 #1222]  INFO -- : Writing
/tmp/build_49ba6c877f5502cd4029406e981f90b4/public/assets/application-
1bf5315c71171ad5f9cbef00193d56b7e45263ddc64caf676ce988cfbb6570bd.js
remote:        I, [2016-07-08T17:08:57.046951 #1222]  INFO -- : Writing
```

```
/tmp/build_49ba6c877f5502cd4029406e981f90b4/public/assets/application-
1bf5315c71171ad5f9cbef00193d56b7e45263ddc64caf676ce988cfbb6570bd.js.gz
remote:        I, [2016-07-08T17:08:57.060208 #1222]  INFO -- : Writing
/tmp/build_49ba6c877f5502cd4029406e981f90b4/public/assets/application-
e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855.css
remote:        I, [2016-07-08T17:08:57.060656 #1222]  INFO -- : Writing
/tmp/build_49ba6c877f5502cd4029406e981f90b4/public/assets/application-
e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855.css.gz
remote:        Asset precompilation completed (4.06s)
remote:        Cleaning assets
remote:        Running: rake assets:clean
remote:
remote: ###### WARNING:
remote:        No Procfile detected, using the default web server.
remote:        We recommend explicitly declaring how to boot your server process via a
Procfile.
remote:        https://devcenter.heroku.com/articles/ruby-default-web-server
remote:
remote: -----> Discovering process types
remote:        Procfile declares types     -> (none)
remote:        Default types for buildpack -> console, rake, web, worker
remote:
remote: -----> Compressing...
remote:        Done: 29.2M
remote: -----> Launching...
remote:        Released v5
remote:        https://example.herokuapp.com/ deployed to Heroku
remote:
remote: Verifying deploy... done.
To https://git.heroku.com/example.git
 * [new branch]      master -> master
```

Se si utilizza il database nell'applicazione, è necessario migrare manualmente il database eseguendo:

```
$ heroku run rake db:migrate
```

Qualsiasi comando dopo la `heroku run` verrà eseguito su un dyno Heroku. È possibile ottenere una sessione shell interattiva eseguendo:

```
$ heroku run bash
```

Assicurati di avere un banco prova con il tipo di processo Web:

```
$ heroku ps:scale web=1
```

Il comando heroku ps elenca i dynos in esecuzione della tua applicazione:

```
$ heroku ps
=== web (Standard-1X): bin/rails server -p $PORT -e $RAILS_ENV (1)
web.1: starting 2016/07/08 12:09:06 -0500 (~ 2s ago)
```

Ora puoi visitare l'app nel nostro browser con `heroku open` .

```
$ heroku open
```

Heroku ti fornisce un URL Web predefinito nel dominio `herokuapp.com` . Quando sei pronto per aumentare la produzione, puoi aggiungere il tuo dominio personalizzato.

## Gestione degli ambienti di produzione e di staging per un Heroku

Ogni app di Heroku funziona in almeno due ambienti: su Heroku (chiameremo quella produzione) e sulla tua macchina locale (sviluppo). Se più di una persona sta lavorando sull'app, allora hai più ambienti di sviluppo, uno per macchina, di solito. Di solito, ogni sviluppatore avrà anche un ambiente di test per eseguire test. Sfortunatamente, questo approccio si rompe quando gli ambienti diventano meno simili. Windows e Mac, ad esempio, forniscono entrambi ambienti diversi rispetto allo stack Linux su Heroku, quindi non si può essere sempre sicuri che il codice che funziona nell'ambiente di sviluppo locale funzioni allo stesso modo quando lo si distribuisce in produzione.

La soluzione è avere un ambiente di staging che sia il più simile alla produzione possibile. Questo può essere ottenuto creando una seconda applicazione Heroku che ospita la tua applicazione di staging. Con la messa in scena, è possibile controllare il codice in un'impostazione di tipo produzione prima di averlo influenzato sugli utenti effettivi.

**Partendo da zero**

Supponiamo che tu abbia un'applicazione in esecuzione sul tuo computer locale e sei pronto per inviarla a Heroku. Dovremo creare ambienti remoti, allestimenti e produzione. Per prendere l'abitudine di spingere per la prima messa in scena, inizieremo con questo:

```
$ heroku create --remote staging
Creating strong-river-216.... done
http://strong-river-216.heroku.com/ | https://git.heroku.com/strong-river-216.git
Git remote staging added
```

Per impostazione predefinita, la CLI heroku crea progetti con un heroku git remote. Qui, stiamo specificando un nome diverso con il flag --remote, quindi spingendo il codice su Heroku e eseguendo i comandi contro l'app sembra un po 'diverso dal normale guru di push her goku:

```
 $ git push staging master
...
 $ heroku ps --remote staging
=== web: `bundle exec puma -C config/puma.rb``
web.1: up for 21s
```

Una volta che l'app di gestione temporanea è stata installata correttamente, puoi creare la tua app di produzione:

```
$ heroku create --remote production
Creating fierce-ice-327.... done
http://fierce-ice-327.heroku.com/ | https://git.heroku.com/fierce-ice-327.git
Git remote production added
$ git push production master
```

```
...
$ heroku ps --remote production
=== web: `bundle exec puma -C config/puma.rb
web.1: up for 16s
```

E con questo, hai la stessa base di codice in esecuzione come due app di Heroku separate: una di scena e una di produzione, configurate in modo identico. Ricorda che dovrai specificare quale app utilizzerai per il tuo lavoro quotidiano. Puoi usare flag '--remote' o usare git config per specificare un'app predefinita.

Leggi Distribuzione di un'app Rails su Heroku online: https://riptutorial.com/it/ruby-on-rails/topic/4485/distribuzione-di-un-app-rails-su-heroku

# Capitolo 29: elasticsearch

## Examples

### Installazione e test

La prima cosa che vuoi fare per lo sviluppo locale è installare ElasticSearch nel tuo computer e testarlo per vedere se è in esecuzione. Richiede l'installazione di Java. L'installazione è piuttosto semplice:

- Mac OS X: `brew install elasticsearch`
- Ubuntu: `sudo apt-get install elasticsearch`

Quindi avvialo:

- Mac OS X: i `brew services start elasticsearch`
- Ubuntu: `sudo service elasticsearch start`

Per provarlo, il modo più semplice è con `curl` . Potrebbero essere necessari alcuni secondi per l'avvio, quindi non farti prendere dal panico se non ricevi alcuna risposta all'inizio.

```
curl localhost:9200
```

Risposta di esempio:

```
{
  "name" : "Hydro-Man",
  "cluster_name" : "elasticsearch_gkbonetti",
  "version" : {
    "number" : "2.3.5",
    "build_hash" : "90f439ff60a3c0f497f91663701e64ccd01edbb4",
    "build_timestamp" : "2016-07-27T10:36:52Z",
    "build_snapshot" : false,
    "lucene_version" : "5.5.0"
  },
  "tagline" : "You Know, for Search"
}
```

### Impostazione di strumenti per lo sviluppo

Quando si inizia con ElasticSearch (ES), potrebbe essere utile disporre di uno strumento grafico che consente di esplorare i dati. Un plugin chiamato `elasticsearch-head` fa proprio questo. Per installarlo, fai quanto segue:

- Scopri in quale cartella ES è installato: `ls -l $(which elasticsearch)`
- `cd` in questa cartella ed esegui l'installazione plugin binary: `elasticsearch/bin/plugin -install mobz/elasticsearch-head`
- Aprire `http://localhost:9200/_plugin/head/` nel browser

Se tutto ha funzionato come previsto, dovresti vedere una bella GUI in cui esplorare i tuoi dati.

## introduzione

ElasticSearch ha un'API JSON ben documentata, ma probabilmente vorrai utilizzare alcune librerie che gestiscono ciò per te:

- `Elasticsearch` : il wrapper ufficiale di basso livello per l'API HTTP
- `Elasticsearch-rails` : l'integrazione ufficiale di rails di alto livello che ti aiuta a connettere i tuoi modelli Rails con ElasticSearch usando il pattern ActiveRecord o Repository
- `Chewy` - `Chewy` Rails alternativa e non ufficiale di alto livello che è molto popolare e probabilmente ha una documentazione migliore

Usiamo la prima opzione per testare la connessione:

```
gem install elasticsearch
```

Quindi accendi il terminale rubino e provalo:

```
require 'elasticsearch'

client = Elasticsearch::Client.new log: true
# by default it connects to http://localhost:9200

client.transport.reload_connections!
client.cluster.health

client.search q: 'test'
```

## Searchkick

Se vuoi impostare rapidamente elasticsearch puoi utilizzare la searchkick gem:

```
gem 'searchkick'
```

Aggiungi searchkick ai modelli che vuoi cercare.

```
class Product < ActiveRecord::Base
  searchkick
end
```

Aggiungi dati all'indice di ricerca.

```
Product.reindex
```

E per interrogare, usa:

```
products = Product.search "apples"
products.each do |product|
  puts product.name
end
```

Abbastanza veloce, la conoscenza elasticsearch non è richiesta ;-)

Maggiori informazioni qui: https://github.com/ankane/searchkick

Leggi elasticsearch online: https://riptutorial.com/it/ruby-on-rails/topic/6500/elasticsearch

# Capitolo 30: Ereditarietà di una tabella singola

## introduzione

La STI (Single Table Inheritance) è un modello di progettazione basato sull'idea di salvare i dati di più modelli che ereditano tutti dallo stesso modello Base in un'unica tabella nel database.

## Examples

### Esempio di base

Per prima cosa abbiamo bisogno di una tabella per contenere i nostri dati

```
class CreateUsers < ActiveRecord::Migration
  def change
    create_table :users do |t|
      t.string :name
      t.string :password
      t.string :type # <- This makes it an STI

      t.timestamps
    end
  end
end
```

Quindi creiamo alcuni modelli

```
class User < ActiveRecord::Base
   validates_presence_of :password
   # This is a parent class. All shared logic goes here
end

class Admin < User
   # Admins must have more secure passwords than regular users
   # We can add it here
   validates :custom_password_validation
end

class Guest < User
   # Lets say that we have a guest type login.
   # It has a static password that cannot be changed
   validates_inclusion_of :password, in: ['guest_password']
end
```

Quando `Guest.create(name: 'Bob')` un `Guest.create(name: 'Bob')` ActiveRecord lo tradurrà per creare una voce nella tabella Utenti con `type: 'Guest'` .

Quando recuperi il record `bob = User.where(name: 'Bob').first` l'oggetto restituito sarà un'istanza di `Guest` , che può essere trattata con la forza come Utente con `bob.becomes(User)`

diventa è più utile quando si ha a che fare con partial condivisi o route / controller della superclasse anziché della sottoclasse.

### Colonna ereditaria personalizzata

Per impostazione predefinita, il nome della classe del modello STI è memorizzato in una colonna di nome `type` . Ma il suo nome può essere cambiato sovrascrivendo il valore `inheritance_column` in una classe base. Per esempio:

```
class User < ActiveRecord::Base
  self.inheritance_column = :entity_type # can be string as well
end

class Admin < User; end
```

La migrazione in questo caso avrà il seguente aspetto:

```
class CreateUsers < ActiveRecord::Migration
  def change
    create_table :users do |t|
      t.string :name
      t.string :password
      t.string :entity_type

      t.timestamps
    end
  end
end
```

Quando `Admin.create` , questo record verrà salvato nella tabella utenti con `entity_type = "Admin"`

### Modello di rotaie con colonna di tipo e senza STI

Avendo `type` colonna in un modello Rails senza invocare STI può essere ottenuto assegnando `:_type_disabled` a `inheritance_column` :

```
class User < ActiveRecord::Base
  self.inheritance_column = :_type_disabled
end
```

Leggi Ereditarietà di una tabella singola online: https://riptutorial.com/it/ruby-on-rails/topic/9125/ereditarieta-di-una-tabella-singola

# Capitolo 31: Formare aiutanti

## introduzione

Rails fornisce help di visualizzazione per la generazione di markup di modulo.

## Osservazioni

- I tipi di data di ingresso, tra cui `date` , `datetime` , `datetime-local` , `time` , `month` e `week` non funzionano in FireFox.
- `input<type="telephone">` funziona solo con Safari 8.
- `input<type="email">` non funziona su Safari

## Examples

### Crea un modulo

Puoi creare un modulo usando l'helper `form_tag`

```
<%= form_tag do %>
  Form contents
<% end %>
```

Questo crea il seguente codice HTML

```
<form accept-charset="UTF-8" action="/" method="post">
  <input name="utf8" type="hidden" value="&#x2713;" />
  <input name="authenticity_token" type="hidden"
value="J7CBxfHalt49OSHp27hblqK20c9PgwJ108nDHX/8Cts=" />
  Form contents
</form>
```

Questo tag modulo ha creato un campo di input `hidden` . Questo è necessario, perché i moduli non possono essere inviati correttamente senza di esso.

Il secondo campo di input, denominato `authenticity_token` aggiunge protezione contro `cross-site request forgery` .

### Creazione di un modulo di ricerca

Per creare un modulo di ricerca, inserire il seguente codice

```
<%= form_tag("/search", method: "get") do %>
  <%= label_tag(:q, "Search for:") %>
  <%= text_field_tag(:q) %>
  <%= submit_tag("Search") %>
<% end %>
```

- `form_tag` : questo è l'helper predefinito per la creazione di un modulo. È il primo parametro, `/search` è l'azione e il secondo parametro specifica il metodo HTTP. Per i moduli di ricerca, è importante utilizzare sempre il metodo `get`
- `label_tag` : questo helper crea un tag `<label>` html.
- `text_field_tag` : questo creerà un elemento di input con `text` tipo
- `submit_tag` : crea un elemento di input con tipo `submit`

### Aiutanti per gli elementi del modulo

## caselle di controllo

```
<%= check_box_tag(:pet_dog) %>
<%= label_tag(:pet_dog, "I own a dog") %>
<%= check_box_tag(:pet_cat) %>
<%= label_tag(:pet_cat, "I own a cat") %>
```

Questo genererà il seguente codice HTML

```
<input id="pet_dog" name="pet_dog" type="checkbox" value="1" />
<label for="pet_dog">I own a dog</label>
<input id="pet_cat" name="pet_cat" type="checkbox" value="1" />
<label for="pet_cat">I own a cat</label>
```

## Tasti della radio

```
<%= radio_button_tag(:age, "child") %>
<%= label_tag(:age_child, "I am younger than 18") %>
<%= radio_button_tag(:age, "adult") %>
<%= label_tag(:age_adult, "I'm over 18") %>
```

Questo genera il seguente codice HTML

```
<input id="age_child" name="age" type="radio" value="child" />
<label for="age_child">I am younger than 18</label>
<input id="age_adult" name="age" type="radio" value="adult" />
<label for="age_adult">I'm over 18</label>
```

## Area di testo

Per creare una casella di testo più grande, si consiglia di utilizzare `text_area_tag`

```
<%= text_area_tag(:message, "This is a longer text field", size: "25x6") %>
```

Questo creerà il seguente codice HTML

```
<textarea id="message" name="message" cols="25" rows="6">This is a longer text
field</textarea>
```

## Campo numerico

Questo creerà un elemento di `input<type="number">`

```
<%= number_field :product, :rating %>
```

Per specificare un intervallo di valori, possiamo usare l'opzione `in:`

```
<%= number_field :product, :rating, in: 1..10 %>
```

## Campo password

A volte vuoi che i caratteri digitati dall'utente vengano mascherati. Questo genererà un `<input type="password">`

```
<%= password_field_tag(:password) %>
```

## Campo email

Questo creerà un `<input type="email">`

```
<%= email_field(:user, :email) %>
```

## Campo telefonico

Questo creerà un `<input type="tel">` .

```
<%= telephone_field :user, :phone %>
```

## Aiutanti di data

- `input[type="date"]`

  ```
  <%= date_field(:user, :reservation) %>
  ```

- `input[type="week"]`

  ```
  <%= week_field(:user, :reservation) %>
  ```

- `input[type="year"]`

  ```
  <%= year_field(:user, :reservation) %>
  ```

- `input[type="time"]`

```
<%= time_field(:user, :check_in) %>
```

## Cadere in picchiata

Esempio standard: @models = Model.all select_tag "models", options_from_collection_for_select (@models, "id", "name"), {}

Questo genererà il seguente codice HTML: David

L'ultimo argomento sono opzioni, che accettano quanto segue: {multiple: false, disabled: false, include_blank: false, prompt: false}

Altri esempi possono essere trovati:
http://apidock.com/rails/ActionView/Helpers/FormTagHelper/select_tag

Leggi Formare aiutanti online: https://riptutorial.com/it/ruby-on-rails/topic/4509/formare-aiutanti

# Capitolo 32: Funzione di pagamento in binari

## introduzione

Questo documento pretende di presentarti, con un esempio completo, come puoi implementare diversi metodi di pagamento con Ruby on Rails.

Nell'esempio, riguarderemo Stripe e Braintree due piattaforme di pagamento molto conosciute.

## Osservazioni

Documentazione.

Banda

Braintree

## Examples

### Come integrarsi con Stripe

Aggiungi la gemma Stripe al nostro `Gemfile`

```
gem 'stripe'
```

Aggiungi i file `initializers/stripe.rb` . Questo file contiene le chiavi necessarie per la connessione con il tuo account stripe.

```
require 'require_all'

Rails.configuration.stripe = {
    :publishable_key => ENV['STRIPE_PUBLISHABLE_KEY'],
    :secret_key      => ENV['STRIPE_SECRET_KEY']
}

Stripe.api_key = Rails.configuration.stripe[:secret_key]
```

# Come creare un nuovo cliente a Stripe

```
Stripe::Customer.create({email: email, source: payment_token})
```

Questo codice crea un nuovo cliente su Stripe con un determinato indirizzo email e origine.

`payment_token` è il token dato dal lato client che contiene un metodo di pagamento come una carta di credito o un conto bancario. Ulteriori informazioni: Stripe.js lato client

# Come recuperare un piano da Stripe

```
Stripe::Plan.retrieve(stripe_plan_id)
```

Questo codice recupera un piano da Stripe tramite il suo id.

# Come creare un abbonamento

Quando abbiamo un cliente e un piano possiamo creare un nuovo abbonamento su Stripe.

```
Stripe::Subscription.create(customer: customer.id, plan: plan.id)
```

Creerà un nuovo abbonamento e addebiterà il nostro utente. È importante sapere che cosa realmente accade su Stripe quando iscriviamo un utente a un piano, troverai maggiori informazioni qui: Ciclo di vita dell'iscrizione a strisce .

# Come caricare un utente con un singolo pagamento

A volte vogliamo addebitare ai nostri utenti una sola volta, per fare ciò faremo il prossimo.

```
Stripe::Charge.create(amount:   amount, customer: customer, currency: currency)
```

In tal caso, addebitiamo il nostro utente una volta per un determinato importo.

Errori comuni:

- L'importo deve essere inviato in forma intera, ciò significa che il 2000 sarà di 20 unità di valuta. Controlla questo esempio
- Non è possibile caricare un utente in due valute. Se l'utente è stato addebitato in EUR in qualsiasi momento in passato, non è possibile addebitare l'importo in USD.
- Non è possibile caricare l'utente senza fonte (metodo di pagamento).

Leggi Funzione di pagamento in binari online: https://riptutorial.com/it/ruby-on-rails/topic/10929/funzione-di-pagamento-in-binari

# Capitolo 33: Gems

## Osservazioni

## Documentazione Gemfile

Per i progetti che dovrebbero crescere, è una buona idea aggiungere commenti al tuo `Gemfile` . In questo modo, anche in configurazioni di grandi dimensioni, saprai ancora cosa fa ogni gemma, anche se il nome non è auto-esplicativo e lo hai aggiunto 2 anni fa.

Questo può anche aiutarti a ricordare perché hai scelto una determinata versione e di conseguenza rivalutare in seguito i requisiti della versione.

Esempi:

```
# temporary downgrade for TeamCity
gem 'rake', '~> 10.5.0'
# To upload invoicing information to payment provider
gem 'net-sftp'
```

## Examples

### Cos'è una gemma?

Una gemma è l'equivalente di un plugin o un'estensione per il linguaggio di programmazione ruby.

Per essere precisi anche i binari non sono altro che una gemma. Molte gemme sono costruite su rotaie o altre gemme (dipendono dalla gemma) o sono indipendenti.

# Nel tuo progetto Rails

### Gemfile

Per il tuo progetto Rails hai un file chiamato `Gemfile` . Qui puoi aggiungere le gemme che desideri includere e utilizzare nel tuo progetto. Una volta aggiunto, è necessario installare la gemma usando il `bundler` (vedere la sezione Bundler).

### Gemfile.lock

Una volta fatto questo, il tuo `Gemfile.lock` verrà aggiornato con le gemme appena aggiunte e le loro dipendenze. Questo file blocca le gemme utilizzate in modo che utilizzino quella versione specifica dichiarata in quel file.

```
GEM
```

```
remote: https://rubygems.org/
specs:
devise (4.0.3)
bcrypt (~> 3.0)
orm_adapter (~> 0.1)
railties (>= 4.1.0, < 5.1)
responders
warden (~> 1.2.3)
```

Questo esempio è per la gemma `devise` . Nel file `Gemfile.lock` viene dichiarata la versione `4.0.3` da indicare quando si installa il progetto su un'altra macchina o sul server di produzione con la versione specificata da utilizzare.

# Sviluppo

O una singola persona, un gruppo o un'intera comunità lavora e mantiene una gemma. Il lavoro svolto è di solito rilasciato dopo alcuni `issues` sono stati risolti o `features` sono state aggiunte.

Di solito le versioni seguono il principio di Semantic Versioning 2.0.0 .

## Bundler

Il modo più semplice per gestire e gestire le gemme è utilizzare il `bundler` . Bundler è un gestore di pacchetti paragonabile a Bower.

Per usare bundler devi prima installarlo.

```
gem install bundler
```

Dopo aver installato ed eseguito il bundler, tutto ciò che devi fare è aggiungere gemme al tuo `Gemfile` ed eseguire

```
bundle
```

nel tuo terminale. Questo installa le gemme appena aggiunte al tuo progetto. In caso di problemi, riceverai un messaggio nel tuo terminale.

Se sei interessato a maggiori dettagli, ti suggerisco di dare un'occhiata ai documenti .

## Gemfiles

Per iniziare, i gemfile richiedono almeno un'origine, sotto forma di URL per un server RubyGems.

Genera un Gemfile con il sorgente rubygems.org predefinito eseguendo il `bundle init` . Usa https in modo che la tua connessione al server venga verificata con SSL.

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

Quindi, dichiara le gemme di cui hai bisogno, compresi i numeri di versione.

```
gem 'rails', '4.2.6'
gem 'rack',  '>=1.1'
gem 'puma',  '~>3.0'
```

La maggior parte degli specificatori di versione, come> = 1.0, sono autoesplicativi. Lo specificatore ~> ha un significato speciale. ~> 2.0.3 è identico a> = 2.0.3 e <2.1. ~> 2.1 è identico a> = 2.1 e <3.0. ~> 2.2.beta corrisponderà alle versioni prerelease come 2.2.beta.12.

I repository Git sono anche fonti gemme valide, purché il repository contenga una o più gemme valide. Specificare cosa controllare con `:tag` `:branch` , o `:ref` . L'impostazione predefinita è il ramo `master` .

```
gem 'nokogiri', :git => 'https://github.com/sparklemotion/nokogiri', :branch => 'master'
```

Se desideri utilizzare una gemma decompressa direttamente dal filesystem, imposta semplicemente l'opzione: path al percorso contenente i file della gem.

```
gem 'extracted_library', :path => './vendor/extracted_library'
```

Le dipendenze possono essere inserite in gruppi. I gruppi possono essere ignorati al momento dell'installazione (usando `--without` ) o richiesti contemporaneamente (usando `Bundler.require` ).

```
gem 'rails_12factor', group: :production

group :development, :test do
  gem 'byebug'
  gem 'web-console', '~> 2.0'
  gem 'spring'
  gem 'dotenv-rails'
end
```

Puoi specificare la versione richiesta di Ruby nel Gemfile con `ruby` . Se il Gemfile è caricato su una diversa versione di Ruby, Bundler solleverà un'eccezione con una spiegazione.

```
ruby '2.3.1'
```

## Gemsets

Se stai usando `RVM(Ruby Version Manager)` allora usare un `gemset` per ogni progetto è una buona idea. Un `gemset` è solo un contenitore che puoi usare per tenere le gemme separate l'una dall'altra. La creazione di un `gemset` per progetto ti consente di cambiare le gemme (e le versioni gem) per un progetto senza rompere tutti gli altri tuoi progetti. Ogni progetto deve solo preoccuparsi delle proprie gemme.

`RVM` fornisce (> = 0.1.8) un interprete `@global gemset` per ruby. Le gemme che installi sul `@global gemset` per un dato rubino sono disponibili per tutti gli altri set di gemme che crei in associazione con quel rubino. Questo è un buon modo per consentire a tutti i tuoi progetti di condividere lo

stesso gioi installato per una specifica installazione dell'interprete ruby.

**Creazione di gemme**

Supponiamo che tu abbia già installato `ruby-2.3.1` e tu lo abbia selezionato usando questo comando:

```
rvm use ruby-2.3.1
```

Ora per creare gemset per questa versione rubino:

```
rvm gemset create new_gemset
```

dove `new_gemset` è il nome di gemset. Per vedere l'elenco di gemsets disponibili per una versione ruby:

```
rvm gemset list
```

per elencare le gemme di tutte le versioni di rubino:

```
rvm gemset list_all
```

usare un gemset dalla lista (supponiamo che `new_gemset` sia il gemset che voglio usare):

```
rvm gemset use new_gemset
```

puoi anche specificare la versione rubino con il gemset se vuoi passare ad un'altra versione di rubino:

```
rvm use ruby-2.1.1@new_gemset
```

per specificare un gemset predefinito per una particolare versione ruby:

```
rvm use 2.1.1@new_gemset --default
```

per rimuovere tutte le gemme installate da un gemset puoi svuotarlo tramite:

```
rvm gemset empty new_gemset
```

per copiare un gioiello da un rubino all'altro puoi farlo da:

```
rvm gemset copy 2.1.1@rails4 2.1.2@rails4
```

eliminare un gemset:

```
rvm gemset delete new_gemset
```

per vedere il nome attuale del gemset:

```
rvm gemset name
```

installare una gemma nel gemset globale:

```
rvm @global do gem install ...
```

**Inizializzazione dei set di gemme durante le installazioni di Ruby**

Quando installi un nuovo ruby, RVM non solo crea due gemsets (il default, il gemset vuoto e il gemset globale), ma usa anche una serie di file modificabili dall'utente per determinare quali gemme installare.

Lavorando in `~/.rvm/gemsets` , `~/.rvm/gemsets` cerca `global.gems` e `default.gems` usando una `global.gems` albero in base alla stringa ruby da installare. Utilizzando l'esempio di `ree-1.8.7-p2010.02` , `ree-1.8.7-p2010.02` controllerà (e importerà da) i seguenti file:

```
~/.rvm/gemsets/ree/1.8.7/p2010.02/global.gems
~/.rvm/gemsets/ree/1.8.7/p2010.02/default.gems
~/.rvm/gemsets/ree/1.8.7/global.gems
~/.rvm/gemsets/ree/1.8.7/default.gems
~/.rvm/gemsets/ree/global.gems
~/.rvm/gemsets/ree/default.gems
~/.rvm/gemsets/global.gems
~/.rvm/gemsets/default.gems
```

Ad esempio, se hai modificato `~/.rvm/gemsets/global.gems` aggiungendo queste due righe:

```
bundler
awesome_print
```

ogni volta che installi un nuovo rubino, queste due gemme vengono installate nel tuo gemst globale. `global.gems` file `default.gems` e `global.gems` vengono solitamente sovrascritti durante l'aggiornamento di rvm.

Leggi Gems online: https://riptutorial.com/it/ruby-on-rails/topic/3130/gems

# Capitolo 34: I18n - Internazionalizzazione

## Sintassi

- I18n.t ( "chiave")
- I18n.translate ("chiave") # equivalente a `I18n.t("key")`
- I18n.t ("chiave", conteggio: 4)
- I18n.t ("chiave", param1: "Qualcosa", param2: "Altro")
- I18n.t ("doesnt_exist", default: "chiave") # specifica un valore predefinito se manca la chiave
- I18n.locale # =>: en
- I18n.locale =: en
- I18n.default_locale # =>: en
- I18n.default_locale =: en
- t (". chiave") # uguale a `I18n.t("key")` , ma con l'ambito dell'azione / modello chiamato da

## Examples

### Usa I18n in visualizzazioni

Supponendo che tu abbia questo file locale YAML:

```
# config/locales/en.yml
en:
  header:
    title: "My header title"
```

e vuoi mostrare la stringa del titolo, puoi farlo

```
# in ERB files
<%= t('header.title') %>

# in SLIM files
= t('header.title')
```

### I18n con argomenti

È possibile passare parametri al metodo **I18n** `t` :

```
# Example config/locales/en.yml
en:
  page:
    users: "%{users_count} users currently online"

# In models, controller, etc...
I18n.t('page.users', users_count: 12)

# In views
```

```
# ERB
<%= t('page.users', users_count: 12) %>

#SLIM
= t('page.users', users_count: 12)

# Shortcut in views - DRY!
# Use only the dot notation
# Important: Consider you have the following controller and view page#users

# ERB Example app/views/page/users.html.erb
<%= t('.users', users_count: 12) %>
```

E ottieni il seguente risultato:

```
"12 users currently online"
```

## pluralizzazione

Puoi lasciare che **I18n** gestisca la pluralizzazione per te, basta usare l'argomento `count` .

Devi impostare il tuo file locale in questo modo:

```
# config/locales/en.yml
en:
  online_users:
    one: "1 user is online"
    other: "%{count} users are online"
```

Quindi utilizzare la chiave appena creata passando l'argomento `count I18n.t` :

```
I18n.t("online_users", count: 1)
#=> "1 user is online"

I18n.t("online_users", count: 4)
#=> "4 users are online"
```

## Imposta la locale attraverso le richieste

Nella maggior parte dei casi, potresti voler impostare la locale **`I18n`** . Si potrebbe desiderare di impostare le impostazioni internazionali per la sessione corrente, l'utente corrente o in base a un parametro URL. Ciò è facilmente ottenibile implementando un'azione `before_action` in uno dei controller o in `ApplicationController` per averlo in tutti i controller.

```
class ApplicationController < ActionController::Base
  before_action :set_locale

  protected

  def set_locale
    # Remove inappropriate/unnecessary ones
    I18n.locale = params[:locale] ||    # Request parameter
      session[:locale] ||               # Current session
```

```
      (current_user.preferred_locale if user_signed_in?) ||  # Model saved configuration
      extract_locale_from_accept_language_header ||          # Language header - browser
config
      I18n.default_locale               # Set in your config files, english by super-default
  end

  # Extract language from request header
  def extract_locale_from_accept_language_header
    if request.env['HTTP_ACCEPT_LANGUAGE']
      lg = request.env['HTTP_ACCEPT_LANGUAGE'].scan(/^[a-z]{2}/).first.to_sym
      lg.in?([:en, YOUR_AVAILABLE_LANGUAGES]) ? lg : nil
    end
  end
```

# basata URL

Il parametro `locale` potrebbe provenire da un URL come questo

```
http://yourapplication.com/products?locale=en
```

O

```
http://yourapplication.com/en/products
```

Per raggiungere quest'ultimo, è necessario modificare i `routes` , aggiungendo un `scope` :

```
# config/routes.rb
scope "(:locale)", locale: /en|fr/ do
  resources :products
end
```

In questo modo, visitando `http://yourapplication.com/en/products` verranno impostate le impostazioni internazionali su `:en` . Invece, visitando `http://yourapplication.com/fr/products` lo imposterai a `:fr` . Inoltre, non si ottiene un errore di routing quando manca il parametro `:locale` , poiché visitando `http://yourapplication.com/products` verrà caricata la locale **I18n** predefinita.

# Basato su sessione o basato sulla persistenza

Ciò presuppone che l'utente possa fare clic su un pulsante / flag lingua per cambiare la lingua. L'azione può essere instradata a un controller che imposta la sessione sulla lingua corrente (e eventualmente persiste le modifiche in un database se l'utente è connesso)

```
class SetLanguageController < ApplicationController
  skip_before_filter :authenticate_user!
  after_action :set_preferred_locale
```

```
  # Generic version to handle a large list of languages
  def change_locale
    I18n.locale = sanitize_language_param
    set_session_and_redirect
  end
```

Devi definire sanitize_language_param con l'elenco delle lingue disponibili e infine gestire gli errori nel caso in cui la lingua non esista.

Se hai pochissime lingue, potrebbe valere la pena di definirle in questo modo:

```
def fr
  I18n.locale = :fr
  set_session_and_redirect
end

def en
  I18n.locale = :en
  set_session_and_redirect
end

private

  def set_session_and_redirect
    session[:locale] = I18n.locale
    redirect_to :back
  end

  def set_preferred_locale
    if user_signed_in?
      current_user.preferred_locale = I18n.locale.to_s
      current_user.save if current_user.changed?
    end
  end
end
```

*Nota: non dimenticare di aggiungere alcuni percorsi alle tue azioni* `change_language`

# Impostazioni internazionali predefinite

Ricordare che è necessario impostare le impostazioni locali predefinite dell'applicazione. Puoi farlo impostandolo in `config/application.rb` :

```
config.i18n.default_locale = :de
```

o creando un inizializzatore nella cartella `config/initializers` :

```
# config/initializers/locale.rb
I18n.default_locale = :it
```

## Ottieni impostazioni locali dalla richiesta HTTP

A volte può essere utile impostare le impostazioni locali dell'applicazione in base all'IP della richiesta. Puoi facilmente ottenerlo usando **`Geocoder`** . Tra le molte cose che fa **`Geocoder`** , può anche indicare la `location` di una `request` .

Per prima cosa, aggiungi **`Geocoder`** al tuo `Gemfile`

```
# Gemfile
gem 'geocoder'
```

**`Geocoder`** aggiunge i metodi `location` e `safe_location` standard `Rack::Request` modo da poter facilmente cercare la posizione di qualsiasi richiesta HTTP per indirizzo IP. È possibile utilizzare questi metodi in un'azione `before_action` in `ApplicationController` :

```
class ApplicationController < ActionController::Base
  before_action :set_locale_from_request

  def set_locale_from_request
    country_code = request.location.data["country_code"] #=> "US"
    country_sym = country_code.underscore.to_sym #=> :us

    # If request locale is available use it, otherwise use I18n default locale
    if I18n.available_locales.include? country_sym
      I18n.locale = country_sym
    end
end
```

Attenzione che questo non funzionerà negli ambienti di `development` e `test` , poiché cose come `0.0.0.0` e `localhost` sono validi indirizzi IP Internet validi.

---

# Limitazioni e alternative

**`Geocoder`** è molto potente e flessibile, ma deve essere configurato per funzionare con un *servizio di geocoding* (vedi maggiori dettagli ); molti dei quali pongono limiti all'uso. Vale anche la pena ricordare che chiamare un servizio esterno a ogni richiesta potrebbe influire sulle prestazioni.

Per affrontarli, può anche valere la pena considerare:

## 1. Una soluzione offline

L'utilizzo di una gemma come **`GeoIP`** (vedi qui ) consente di effettuare ricerche su un file di dati locale. Ci può essere un trade-off in termini di accuratezza, dal momento che questi file di dati devono essere mantenuti aggiornati.

## 2. Usa CloudFlare

Le pagine pubblicate tramite CloudFlare hanno la possibilità di essere geocodificate in modo trasparente, con l'aggiunta del codice paese all'intestazione ( `HTTP_CF_IPCOUNTRY` ). Maggiori dettagli

possono essere trovati qui .

## Tradurre gli attributi del modello ActiveRecord

**`globalize`** gem è un'ottima soluzione per aggiungere traduzioni ai tuoi modelli `ActiveRecord` . Puoi installarlo aggiungendo questo al tuo `Gemfile` :

```
gem 'globalize', '~> 5.0.0'
```

Se stai usando `Rails 5` dovrai anche aggiungere `activemodel-serializers-xml`

```
gem 'activemodel-serializers-xml'
```

Le traduzioni di modelli ti consentono di tradurre i valori degli attributi dei tuoi modelli, ad esempio:

```
class Post < ActiveRecord::Base
  translates :title, :text
end

I18n.locale = :en
post.title # => Globalize rocks!

I18n.locale = :he
post.title # => גלובאלייז 2 שולט!
```

Dopo aver definito gli attributi del modello che devono essere tradotti, è necessario creare una tabella di conversione, attraverso una migrazione. **`globalize`** fornisce `create_translation_table!` e `drop_translation_table!` .

Per questa migrazione è necessario utilizzare `up` e `down` , e **non** `change` . Inoltre, per eseguire correttamente questa migrazione, devi prima definire gli attributi tradotti nel tuo modello, come mostrato sopra. Una migrazione corretta per il modello `Post` precedente è questa:

```
class CreatePostsTranslationTable < ActiveRecord::Migration
  def up
    Post.create_translation_table! title: :string, text: :text
  end

  def down
    Post.drop_translation_table!
  end
end
```

Puoi anche passare opzioni per opzioni specifiche, come:

```
class CreatePostsTranslationTable < ActiveRecord::Migration
  def up
    Post.create_translation_table! title: :string,
      text: { type: :text, null: false, default: "Default text" }
  end

  def down
```

```
    Post.drop_translation_table!
  end
end
```

Se hai già **dati esistenti** nelle tue colonne di traduzione che necessitano, puoi facilmente migrarlo alla tabella delle traduzioni, regolando la tua migrazione:

```
class CreatePostsTranslationTable < ActiveRecord::Migration
  def up
    Post.create_translation_table!({
      title: :string,
      text: :text
    }, {
      migrate_data: true
    })

  end

  def down
    Post.drop_translation_table! migrate_data: true
  end
end
```

***Assicurati di eliminare le colonne tradotte dalla tabella padre dopo che tutti i tuoi dati sono stati migrati in modo sicuro.*** Per rimuovere automaticamente le colonne tradotte dalla tabella padre dopo la migrazione dei dati, aggiungi l'opzione `remove_source_columns` alla migrazione:

```
class CreatePostsTranslationTable < ActiveRecord::Migration
  def up
    Post.create_translation_table!({
      title: :string,
      text: :text
    }, {
      migrate_data: true,
      remove_source_columns: true
    })

  end

  def down
    Post.drop_translation_table! migrate_data: true
  end
end
```

Puoi anche aggiungere nuovi campi a una tabella di traduzioni creata in precedenza:

```
class Post < ActiveRecord::Base
  # Remember to add your attribute here too.
  translates :title, :text, :author
end

class AddAuthorToPost < ActiveRecord::Migration
  def up
    Post.add_translation_fields! author: :text
  end
```

```
  def down
    remove_column :post_translations, :author
  end
end
```

## Usa I18n con tag e simboli HTML

```
# config/locales/en.yml
en:
  stackoverflow:
    header:
      title_html: "Use <strong>I18n</strong> with Tags &amp; Symbols"
```

Nota l'aggiunta di `_html` extra dopo il `title` del nome.

E in Views,

```
# ERB
<h2><%= t(:title_html, scope: [:stackoverflow, :header]) %></h2>
```

Leggi I18n - Internazionalizzazione online: https://riptutorial.com/it/ruby-on-rails/topic/2772/i18n---internazionalizzazione

# Capitolo 35: ID amichevole

## introduzione

FriendlyId è il "bulldozer dell'esercito svizzero" di plug-in slugging e permalink per Active Record. Ti consente di creare URL carini e di lavorare con stringhe amichevoli come se fossero id numerici. Con FriendlyId, è facile far sì che la tua applicazione usi URL come:

http://example.com/states/washington

## Examples

### Rails Quickstart

```
rails new my_app
cd my_app
```

# Gemfile

```
gem 'friendly_id', '~> 5.1.0' # Note: You MUST use 5.0.0 or greater for Rails 4.0+
rails generate friendly_id
rails generate scaffold user name:string slug:string:uniq
rake db:migrate
```

# modifica app / models / user.rb

```
class User < ApplicationRecord
  extend FriendlyId
  friendly_id :name, use: :slugged
end

User.create! name: "Joe Schmoe"

# Change User.find to User.friendly.find in your controller
User.friendly.find(params[:id])
```

```
rails server
GET http://localhost:3000/users/joe-schmoe
```

```
# If you're adding FriendlyId to an existing app and need
# to generate slugs for existing users, do this from the
```

```
# console, runner, or add a Rake task:
User.find_each(&:save)

Finders are no longer overridden by default. If you want to do friendly finds, you must do
Model.friendly.find rather than Model.find. You can however restore FriendlyId 4-style finders
by using the :finders addon


friendly_id :foo, use: :slugged # you must do MyClass.friendly.find('bar')
#or...
friendly_id :foo, use: [:slugged, :finders] # you can now do MyClass.find('bar')
```

Una nuova funzionalità "candidati" che semplifica la configurazione di un elenco di slug alternativi che possono essere utilizzati per distinguere in modo univoco i record piuttosto che aggiungere una sequenza. Per esempio:

```
class Restaurant < ActiveRecord::Base
  extend FriendlyId
  friendly_id :slug_candidates, use: :slugged

  # Try building a slug based on the following fields in
  # increasing order of specificity.
  def slug_candidates
    [
      :name,
      [:name, :city],
      [:name, :street, :city],
      [:name, :street_number, :street, :city]
    ]
  end
end
```

Imposta la lunghezza limite di slug usando la gem di friendly_id?

```
def normalize_friendly_id(string)
   super[0..40]
end
```

Leggi ID amichevole online: https://riptutorial.com/it/ruby-on-rails/topic/9664/id-amichevole

# Capitolo 36: Importa interi file CSV da una cartella specifica

## introduzione

In questo esempio, diciamo che abbiamo molti file CSV del prodotto in una cartella. Ogni file CSV deve caricare il nostro database dalla nostra console scrivere un comando. Eseguire il seguente comando in un progetto nuovo o esistente per creare questo modello.

## Examples

### Carica CSV dal comando della console

Comandi del terminale:

```
rails g model Product name:string quantity:integer price:decimal{12,2}
rake db:migrate
```

Lates crea controller.

Comandi del terminale:

```
rails g controller Products
```

Codice controller:

```
class HistoriesController < ApplicationController
    def create
        file = Dir.glob("#{Rails.root}/public/products/**/*.csv") #=> This folder directory
where read the CSV files
        file.each do |file|
            Product.import(file)
        end
    end
end
```

Modello:

```
class Product< ApplicationRecord
  def self.import(file)
      CSV.foreach(file.path, headers: true) do |row|
          Product.create! row.to_hash
      end
  end
end
```

routes.rb

```
resources :products
```

app / config / application.rb

```
require 'csv'
```

Ora apri la tua `console` sviluppo ed `run`

```
=> ProductsController.new.create #=> Uploads your whole CSV files from your folder directory
```

Leggi Importa interi file CSV da una cartella specifica online: https://riptutorial.com/it/ruby-on-rails/topic/8658/importa-interi-file-csv-da-una-cartella-specifica

# Capitolo 37: Integrazione di React.js con Rails mediante Hyperloop

## introduzione

Questo argomento riguarda l'integrazione di React.js con Rails utilizzando la gemma Hyperloop

Altri approcci non trattati qui utilizzano i reatt-rails o react_on_rails gems.

## Osservazioni

Le classi di componenti generano semplicemente le classi di componenti javascript equivalenti.

Puoi anche accedere ai componenti e alle librerie javascript direttamente dalle classi dei componenti del ruby.

Hyperloop "eseguirà il prerender" del lato server di visualizzazione in modo che la vista iniziale venga caricata proprio come i modelli ERB o HAML. Una volta caricato sul client, la reazione riprende e aggiorna in modo incrementale il DOM come cambiamenti di stato dovuti agli input dell'utente, alle richieste HTTP o ai dati del socket Web in ingresso.

Oltre ai componenti, Hyperloop ha negozi per gestire lo stato condiviso, operazioni per incapsulare la logica aziendale isomorfa e modelli che danno accesso diretto ai modelli ActiveRecord sul client utilizzando la sintassi AR standard.

Maggiori informazioni qui: http://ruby-hyperloop.io/

## Examples

### Aggiunta di un componente semplice (scritto in rubino) alla tua app Rails

1. Aggiungi il gioiello hyperloop ai tuoi binari (4.0 - 5.1) Gemfile
2. `bundle install`
3. Aggiungi il manifest hyperloop al file application.js:

```
// app/assets/javascripts/application.js
...
//= hyperloop-loader
```

4. Crea i tuoi componenti di reazione e inseriscili nella directory `hyperloop/components`

```
# app/hyperloop/components/hello_world.rb
class HelloWorld < Hyperloop::Component
  after_mount do
    every(1.second) { mutate.current_time(Time.now) }
  end
```

```
  render do
    "Hello World!  The time is now: #{state.current_time}"
  end
end
```

5. I componenti funzionano come le visualizzazioni. Sono "montati" usando il metodo `render_component` in un controller:

```
# somewhere in a controller:
  ...
  def hello_world
    render_component # renders HelloWorld based on method name
  end
```

## Dichiarazione dei parametri dei componenti (oggetti di scena)

```
class Hello < Hyperloop::Component
  # params (= react props) are declared using the param macro
  param :guest
  render do
    "Hello there #{params.guest}"
  end
end

# to "mount" Hello with guest = "Matz" say
  Hello(guest: 'Matz')

# params can be given a default value:
  param guest: 'friend' # or
  param :guest, default: 'friend'
```

## Tag HTML

```
# HTML tags are built in and are UPCASE
class HTMLExample < Hyperloop::Component
  render do
    DIV do
      SPAN { "Hello There" }
      SPAN { "Welcome to the Machine!" }
    end
  end
end
```

## Gestori di eventi

```
# Event handlers are attached using the 'on' method
class ClickMe < Hyperloop::Component
  render do
    DIV do
      SPAN { "Hello There" }
      A { "Click Me" }.on(:click) { alert('you did it!' }
    end
  end
end
```

## stati

```
# States are read using the 'state' method, and updated using 'mutate'
# when states change they cause re-render of all dependent dom elements

class StateExample < Hyperloop::Component
  state count: 0  # by default states are initialized to nil
  render do
    DIV do
      SPAN { "Hello There" }
      A { "Click Me" }.on(:click) { mutate.count(state.count + 1) }
      DIV do
        "You have clicked me #{state.count} #{'time'.pluralize(state.count)}"
      end unless state.count == 0
    end
  end
end
```

Si noti che gli stati possono essere condivisi tra i componenti utilizzando Hyperloop :: Stores

## callback

```
# all react callbacks are supported using active-record-like syntax

class SomeCallBacks < Hyperloop::Component
  before_mount do
    # initialize stuff - replaces normal class initialize method
  end
  after_mount do
    # any access to actual generated dom node, or window behaviors goes here
  end
  before_unmount do
    # any cleanups (i.e. cancel intervals etc)
  end

  # you can also specify a method the usual way:
  before_mount :do_some_more_initialization
end
```

Leggi Integrazione di React.js con Rails mediante Hyperloop online: https://riptutorial.com/it/ruby-on-rails/topic/9809/integrazione-di-react-js-con-rails-mediante-hyperloop

# Capitolo 38: Interfaccia di query ActiveRecord

## introduzione

ActiveRecord è la M in MVC, che è il livello del sistema responsabile della rappresentazione dei dati e della logica aziendale. La tecnica che collega gli oggetti ricchi di un'applicazione alle tabelle in un sistema di gestione di database relazionali è **O** bject **R** elational **M** apper ( **ORM** ).

ActiveRecord eseguirà query sul database per te ed è compatibile con la maggior parte dei sistemi di database. Indipendentemente dal sistema di database che stai utilizzando, il formato del metodo ActiveRecord sarà sempre lo stesso.

## Examples

### .dove

Il metodo `where` è disponibile su qualsiasi modello `ActiveRecord` e consente di eseguire query sul database per un set di record che soddisfano i criteri specificati.

Il metodo `where` accetta un hash in cui le chiavi corrispondono ai nomi delle colonne sulla tabella rappresentata dal modello.

Come semplice esempio, utilizzeremo il seguente modello:

```
class Person < ActiveRecord::Base
  #attribute :first_name, :string
  #attribute :last_name, :string
end
```

Per trovare tutte le persone con il nome di `Sven` :

```
people = Person.where(first_name: 'Sven')
people.to_sql # "SELECT * FROM people WHERE first_name='Sven'"
```

Per trovare tutte le persone con il nome di `Sven` e il cognome di `Schrodinger` :

```
people = Person.where(first_name: 'Sven', last_name: 'Schrodinger')
people.to_sql # "SELECT * FROM people WHERE first_name='Sven' AND last_name='Schrodinger'"
```

Nell'esempio sopra, l'output sql mostra che i record verranno restituiti solo se sia il `first_name` che il `last_name` coincidono.

#### interrogare con la condizione OR

Per trovare i record con `first_name == 'Bruce'` OR `last_name == 'Wayne'`

```
User.where('first_name = ? or last_name = ?', 'Bruce', 'Wayne')
# SELECT "users".* FROM "users" WHERE (first_name = 'Bruce' or last_name = 'Wayne')
```

### .where con un array

Il metodo `where` su qualsiasi modello di ActiveRecord può essere utilizzato per generare SQL del modulo `WHERE column_name IN (a, b, c, ...)` . Questo risultato è ottenuto passando un array come argomento.

Come semplice esempio, utilizzeremo il seguente modello:

```
class Person < ActiveRecord::Base
  #attribute :first_name, :string
  #attribute :last_name, :string
end

people = Person.where(first_name: ['Mark', 'Mary'])
people.to_sql # "SELECT * FROM people WHERE first_name IN ('Mark', 'Mary')"
```

Se la matrice contiene un valore `nil` , lo SQL verrà modificato per verificare se la colonna è `null` :

```
people = Person.where(first_name: ['Mark', 'Mary', nil])
people.to_sql # "SELECT * FROM people WHERE first_name IN ('Mark', 'Mary') OR first_name IS
NULL"
```

### Scopes

Gli ambiti agiscono come filtri predefiniti sui modelli `ActiveRecord` .

Un ambito viene definito utilizzando il metodo della classe `scope` .

Come semplice esempio, utilizzeremo il seguente modello:

```
class Person < ActiveRecord::Base
  #attribute :first_name, :string
  #attribute :last_name, :string
  #attribute :age, :integer

  # define a scope to get all people under 17
  scope :minors, -> { where(age: 0..17) }

  # define a scope to search a person by last name
  scope :with_last_name, ->(name) { where(last_name: name) }

end
```

Gli ambiti possono essere chiamati direttamente dalla classe del modello:

```
minors = Person.minors
```

Gli ambiti possono essere concatenati:

```
peters_children = Person.minors.with_last_name('Peters')
```

Il metodo `where` e altri metodi di tipo query possono anche essere concatenati:

```
mary_smith = Person.with_last_name('Smith').where(first_name: 'Mary')
```

Dietro le quinte, gli ambiti sono semplicemente zucchero sintattico per un metodo di classe standard. Ad esempio, questi metodi sono funzionalmente identici:

```
scope :with_last_name, ->(name) { where(name: name) }

# This ^ is the same as this:

def self.with_last_name(name)
  where(name: name)
end
```

**Ambito di validità**

nel modello per impostare un ambito predefinito per tutte le operazioni sul modello.

> C'è una differenza notevole tra il metodo `scope` e un metodo di classe: gli `scope` definiti da scope restituiranno *sempre* un `ActiveRecord::Relation` , anche se la logica all'interno restituisce nil. I metodi di classe, tuttavia, non hanno una simile rete di sicurezza e possono interrompere la chainability se restituiscono qualcos'altro.

## where.not

`where` clausole possono essere negate usando la sintassi `where.not` :

```
class Person < ApplicationRecord
  #attribute :first_name, :string
end

people = Person.where.not(first_name: ['Mark', 'Mary'])
# => SELECT "people".* FROM "people" WHERE "people"."first_name" NOT IN ('Mark', 'Mary')
```

Supportato da ActiveRecord 4.0 e versioni successive.

## ordinazione

È possibile ordinare i risultati delle query di **ActiveRecord** utilizzando `.order` :

```
User.order(:created_at)
#=> => [#<User id: 2, created_at: "2015-08-12 21:36:23">, #<User id: 11, created_at: "2015-08-
15 10:21:48">]
```

Se non specificato, l'ordine verrà eseguito in ordine crescente. Puoi specificarlo facendo:

```
User.order(created_at: :asc)
```

```
#=> => [#<User id: 2, created_at: "2015-08-12 21:36:23">, #<User id: 11, created_at: "2015-08-
15 10:21:48">]

User.order(created_at: :desc)
#=> [#<User id: 7585, created_at: "2016-07-13 17:15:27">, #<User id: 7583, created_at: "2016-
07-13 16:51:18">]
```

`.order` accetta anche una stringa, quindi puoi anche fare

```
User.order("created_at DESC")
#=> [#<User id: 7585, created_at: "2016-07-13 17:15:27">, #<User id: 7583, created_at: "2016-
07-13 16:51:18">]
```

Poiché la stringa è SQL raw, puoi anche specificare una tabella e non solo un attributo. Supponendo che tu voglia ordinare gli `users` base al loro nome di `role` , puoi farlo:

```
Class User < ActiveRecord::Base
    belongs_to :role
end

Class Role < ActiveRecord::Base
  has_many :users
end

User.includes(:role).order("roles.name ASC")
```

L'ambito `order` può accettare anche un nodo Arel:

```
User.includes(:role).order(User.arel_table[:name].asc)
```

## ActiveRecord Bang (!) Metodi

Se è necessario un metodo **ActiveRecord** per generare un'eccezione anziché un valore `false` in caso di errore, è possibile aggiungere `!` a loro. Questo è molto importante. Dato che alcune eccezioni / fallimenti sono difficili da catturare se non le usi! su di essi. Ho raccomandato di farlo nel tuo ciclo di sviluppo per scrivere tutto il tuo codice ActiveRecord in questo modo per farti risparmiare tempo e fatica.

```
Class User < ActiveRecord::Base
  validates :last_name, presence: true
end

User.create!(first_name: "John")
#=> ActiveRecord::RecordInvalid: Validation failed: Last name can't be blank
```

I metodi **ActiveRecord** che accettano un *botto* ( `!` ) Sono:

- `.create!`
- `.take!`
- `.first!`
- `.last!`
- `.find_by!`

- `.find_or_create_by!`
- `#save!`
- `#update!`
- tutti i cercatori dinamici AR

### .find_by

Puoi trovare i record di qualsiasi campo nella tua tabella usando `find_by` .

Quindi, se hai un modello `User` con un `first_name` puoi fare:

```
User.find_by(first_name: "John")
#=> #<User id: 2005, first_name: "John", last_name: "Smith">
```

`find_by` che `find_by` non genera eccezioni per impostazione predefinita. Se il risultato è un set vuoto, restituisce `nil` invece di `find` .

Se è necessaria l'eccezione, puoi utilizzare `find_by!` che solleva un errore `ActiveRecord::RecordNotFound` come `find` .

### .cancella tutto

Se è necessario eliminare rapidamente molti record, **ActiveRecord** fornisce il metodo **`.delete_all`** . per essere chiamato direttamente su un modello, per cancellare tutti i record in quella tabella o una raccolta. Attenzione però, dato che **`.delete_all`** non istanzia nessun oggetto, quindi non fornisce alcuna callback ( `before_*` e `after_destroy` non vengono attivati).

```
User.delete_all
#=> 39  <-- .delete_all return the number of rows deleted

User.where(name: "John").delete_all
```

### ActiveRecord ricerca tra maiuscole e minuscole

Se è necessario cercare un modello di ActiveRecord per valori simili, si potrebbe essere tentati di utilizzare `LIKE` o `ILIKE` ma questo non è portabile tra i motori di database. Allo stesso modo, il ricorso a downcasing o upcasing sempre in grado di creare problemi di prestazioni.

È possibile utilizzare il metodo di `matches` Arel sottostante di ActiveRecord per farlo in modo sicuro:

```
addresses = Address.arel_table
Address.where(addresses[:address].matches("%street%"))
```

Arel applicherà il costrutto LIKE o ILIKE appropriato per il motore di database configurato.

### Ricevi il primo e l'ultimo record

Le rotaie hanno un modo molto semplice per ottenere il `first` e l' `last` record dal database.

Per ottenere il `first` record dalla tabella degli `users` , dobbiamo digitare il seguente comando:

```
User.first
```

`sql` seguente query `sql` :

```
SELECT  `users`.* FROM `users`  ORDER BY `users`.`id` ASC LIMIT 1
```

E restituirà il seguente record:

```
#<User:0x007f8a6db09920 id: 1, first_name: foo, created_at: Thu, 16 Jun 2016 21:43:03 UTC
+00:00, updated_at: Thu, 16 Jun 2016 21:43:03 UTC +00:00 >
```

Per ottenere l' `last` record dalla tabella degli `users` , dobbiamo digitare il seguente comando:

```
User.last
```

`sql` seguente query `sql` :

```
SELECT  `users`.* FROM `users`  ORDER BY `users`.`id` DESC LIMIT 1
```

E restituirà il seguente record:

```
#<User:0x007f8a6db09920 id: 10, first_name: bar, created_at: Thu, 16 Jun 2016 21:43:03 UTC
+00:00, updated_at: Thu, 16 Jun 2016 21:43:03 UTC +00:00 >
```

Il passaggio di un intero al **primo** e **all'ultimo** metodo crea una query **LIMIT** e restituisce una matrice di oggetti.

```
User.first(5)
```

`sql` seguente query `sql` .

```
SELECT  "users".* FROM "users"  ORDER BY "users"."id" ASC LIMIT 5
```

E

```
User.last(5)
```

`sql` seguente query `sql` .

```
SELECT  "users".* FROM "users"  ORDER BY "users"."id" DESC LIMIT 5
```

### .group e .count

Abbiamo un modello di `Product` e vogliamo raggrupparli per `category` .

```
Product.select(:category).group(:category)
```

Questo interrogherà il database come segue:

```
SELECT "product"."category" FROM "product" GROUP BY "product"."category"
```

Assicurati che anche il campo raggruppato sia selezionato. Il raggruppamento è particolarmente utile per il conteggio dell'occorrenza - in questo caso - delle `categories` .

```
Product.select(:category).group(:category).count
```

Come mostra la query, userà il database per il conteggio, che è molto più efficiente, rispetto al recupero di tutti i record e al conteggio nel codice:

```
SELECT COUNT("products"."category") AS count_categories, "products"."category" AS
products_category FROM "products" GROUP BY "products"."category"
```

## .distinct (o .uniq)

Se vuoi rimuovere i duplicati da un risultato, puoi usare `.distinct()` :

```
Customers.select(:country).distinct
```

Questa query il database come segue:

```
SELECT DISTINCT "customers"."country" FROM "customers"
```

`.uniq()` ha lo stesso effetto. Con Rails 5.0 è stato deprecato e verrà rimosso da Rails con la versione 5.1. Il motivo è che la parola `unique` non ha lo stesso significato di distinta e può essere fuorviante. Inoltre `distinct` è più vicino alla sintassi SQL.

## Si unisce

`joins()` ti permette di unire le tabelle al tuo modello attuale. Per es.

```
User.joins(:posts)
```

produrrà la seguente query SQL:

```
"SELECT "users".* FROM "users" INNER JOIN "posts" ON "posts"."user_id" = "users"."id""
```

Avendo unito la tabella, avrai accesso ad essa:

```
User.joins(:posts).where(posts: { title: "Hello world" })
```

Fai attenzione alla forma plurale. Se la tua relazione è `:has_many` , allora l'argomento `:has_many` `joins()`

dovrebbe essere pluralizzato. Altrimenti, usa il singolare.

Nidificati `joins` :

```
User.joins(posts: :images).where(images: { caption: 'First post' })
```

che produrrà:

```
"SELECT "users".* FROM "users" INNER JOIN "posts" ON "posts"."user_id" = "users"."id" INNER
JOIN "images" ON "images"."post_id" = "images"."id""
```

## include

ActiveRecord with `includes` assicura che tutte le associazioni specificate vengano caricate utilizzando il numero minimo possibile di query. Pertanto, quando si esegue una query su una tabella per i dati con una tabella associata, entrambe le tabelle vengono caricate in memoria.

```
@authors = Author.includes(:books).where(books: { bestseller: true } )

# this will print  results without additional db hitting
@authors.each do |author|
  author.books.each do |book|
    puts book.title
  end
end
```

`Author.joins(:books).where(books: { bestseller: true } )` caricherà solo gli **autori** con condizioni in memoria **senza caricare libri** . Utilizza i `joins` quando non sono richieste informazioni aggiuntive sulle associazioni nidificate.

```
@authors = Author.joins(:books).where(books: { bestseller: true } )

# this will print results without additional queries
@authors.each { |author| puts author.name }

# this will print results with additional db queries
@authors.each do |author|
  author.books.each do |book|
    puts book.title
  end
end
```

## Limite e offset

È possibile utilizzare il `limit` per indicare il numero di record da recuperare e utilizzare l' `offset` per indicare il numero di record da saltare prima di iniziare a restituire i record.

Per esempio

```
User.limit(3) #returns first three records
```

Genererà la seguente query sql.

```
"SELECT  `users`.* FROM `users` LIMIT 3"
```

Poiché l'offset non è menzionato nella query precedente, restituirà i primi tre record.

```
User.limit(5).offset(30) #returns 5 records starting from 31th i.e from 31 to 35
```

Genererà la seguente query sql.

```
"SELECT  `users`.* FROM `users` LIMIT 5 OFFSET 30"
```

Leggi Interfaccia di query ActiveRecord online: https://riptutorial.com/it/ruby-on-rails/topic/2154/interfaccia-di-query-activerecord

# Capitolo 39: Lavori attivi

## Examples

### introduzione

Disponibile da Rails 4.2, Active Job è un framework per la dichiarazione dei lavori e la loro esecuzione su una varietà di backend di accodamento. Le attività ricorrenti o puntuali che non sono bloccanti e possono essere eseguite in parallelo sono buoni casi di utilizzo per Lavori attivi.

### Esempio di lavoro

```
class UserUnsubscribeJob < ApplicationJob
  queue_as :default

  def perform(user)
    # this will happen later
    user.unsubscribe
  end
end
```

### Creazione di un lavoro attivo tramite il generatore

```
$ rails g job user_unsubscribe
```

Leggi Lavori attivi online: https://riptutorial.com/it/ruby-on-rails/topic/8033/lavori-attivi

# Capitolo 40: Logger rotaie

## Examples

### Rails.logger

Usa sempre `Rails.logger.{debug|info|warn|error|fatal}` piuttosto che `puts` . Ciò consente ai registri di adattarsi al formato di log standard, di avere un timestamp e di avere un livello in modo da poter scegliere se sono abbastanza importanti da essere mostrati in un ambiente specifico. Puoi vedere i file di registro separati per la tua applicazione sotto `log/` directory con il nome dell'ambiente dell'app rails. come: `development.log` o `production.log` o `staging.log`

È possibile ruotare facilmente i log di produzione delle guide con LogRotate. Basta eseguire una piccola configurazione come di seguito

Apri `/etc/logrotate.conf` con il tuo editor di linux preferito `vim` o `nano` e aggiungi il codice seguente in questo file in fondo.

```
/YOUR/RAILSAPP/PATH/log/*.log {
  daily
  missingok
  rotate 7
  compress
  delaycompress
  notifempty
  copytruncate
}
```

Quindi, **come funziona** è incredibilmente facile. Ogni bit della configurazione procede come segue:

- **daily** - Ruota i file di registro ogni giorno. Puoi anche utilizzare settimanalmente o mensilmente qui.
- **missingok** - Se il file di registro non esiste, ignorarlo
- **ruotare 7** - Mantenere solo 7 giorni di tronchi
- **comprimi** - GZip il file di registro a rotazione
- **delaycompress** - Ruota il file un giorno, quindi comprimilo il giorno successivo in modo che possiamo essere sicuri che non interferisca con il server Rails
- **notifempty** : non ruotare il file se i registri sono vuoti
- **copytruncate** : copia il file di registro e poi lo svuota. Questo assicura che il file di log Rails stia scrivendo sempre, quindi non avrai problemi perché il file in realtà non cambia. Se non lo usi, dovresti riavviare l'applicazione Rails ogni volta.

**Esecuzione di Logrotate** Poiché abbiamo appena scritto questa configurazione, si desidera testarla.

Per eseguire logrotate manualmente, basta fare: `sudo /usr/sbin/logrotate -f /etc/logrotate.conf`

Questo è tutto.

Leggi Logger rotaie online: https://riptutorial.com/it/ruby-on-rails/topic/3904/logger-rotaie

# Capitolo 41: Memorizzazione sicura delle chiavi di autenticazione

## introduzione

Molte API di terze parti richiedono una chiave che consente loro di prevenire gli abusi. Se ti rilasciano una chiave, è molto importante non impegnare la chiave in un archivio pubblico, in quanto ciò consentirà ad altri di rubare la tua chiave.

## Examples

### Memorizzazione delle chiavi di autenticazione con Figaro

Aggiungi `gem 'figaro'` al tuo Gemfile ed esegui l' `bundle install` . Quindi eseguire `bundle exec figaro install` ; questo creerà config / application.yml e lo aggiungerà al tuo file .gitignore, impedendogli di essere aggiunto al controllo della versione.

È possibile memorizzare le chiavi in application.yml in questo formato:

```
SECRET_NAME: secret_value
```

dove SECRET_NAME e secret_value sono il nome e il valore della chiave API.

Devi anche dare un nome a questi segreti in config / secrets.yml. Puoi avere diversi segreti in ogni ambiente. Il file dovrebbe assomigliare a questo:

```
development:
  secret_name: <%= ENV["SECRET_NAME"] %>
test:
  secret_name: <%= ENV["SECRET_NAME"] %>
production:
  secret_name: <%= ENV["SECRET_NAME"] %>
```

Il modo in cui si utilizzano questi tasti varia, ma per esempio alcuni `some_component` nell'ambiente di sviluppo necessitano dell'accesso a `secret_name` . In config / environments / development.rb, potresti inserire:

```
Rails.application.configure do
  config.some_component.configuration_hash = {
    :secret => Rails.application.secrets.secret_name
  }
end
```

Infine, diciamo che vuoi creare un ambiente di produzione su Heroku. Questo comando caricherà i valori in config / environments / production.rb in Heroku:

```
$ figaro heroku:set -e production
```

Leggi Memorizzazione sicura delle chiavi di autenticazione online: https://riptutorial.com/it/ruby-on-rails/topic/9711/memorizzazione-sicura-delle-chiavi-di-autenticazione

# Capitolo 42: Migrazioni di ActiveRecord

## Parametri

| Tipo di colonna | Descrizione |
|---|---|
| `:primary_key` | Chiave primaria |
| `:string` | Tipo di dati stringa più breve. Consente l'opzione `limit` per il numero massimo di caratteri. |
| `:text` | Quantità di testo più lunga Consente l'opzione `limit` per il numero massimo di byte. |
| `:integer` | Numero intero. Consente l'opzione `limit` per il numero massimo di byte. |
| `:bigint` | Intero più grande |
| `:float` | Galleggiante |
| `:decimal` | Numero decimale con precisione variabile. Permette `precision` e opzioni di `scale` . |
| `:numeric` | Permette `precision` e opzioni di `scale` . |
| `:datetime` | Oggetto DateTime per date / orari. |
| `:time` | Tempo oggetto per tempi. |
| `:date` | Data dell'oggetto per le date. |
| `:binary` | Dati binari Consente l'opzione `limit` per il numero massimo di byte. |
| `:boolean` | booleano |

## Osservazioni

- La maggior parte dei file di migrazione risiede nella directory `db/migrate/` . Sono identificati da un timestamp UTC all'inizio del nome del file: `YYYYMMDDHHMMSS_create_products.rb` .
- Il comando di comando dei `rails generate` può essere abbreviato in `rails g` .
- Se a `:type` non viene passato a un campo, il valore predefinito è una stringa.

## Examples

## Esegui una migrazione specifica

Per eseguire una migrazione specifica verso l'alto o verso il basso, utilizzare `db:migrate:up` o `db:migrate:down` .

Una migrazione specifica:

5.0

```
rake db:migrate:up VERSION=20090408054555
```

5.0

```
rails db:migrate:up VERSION=20090408054555
```

Giù una migrazione specifica:

5.0

```
rake db:migrate:down VERSION=20090408054555
```

5.0

```
rails db:migrate:down VERSION=20090408054555
```

Il numero di versione nei comandi precedenti è il prefisso numerico nel nome file della migrazione. Ad esempio, per migrare alla migrazione `20160515085959_add_name_to_users.rb` , si utilizzerà `20160515085959` come numero di versione.

## Crea una tabella di join

Per creare una tabella di join tra `students` e `courses` , eseguire il comando:

```
$ rails g migration CreateJoinTableStudentCourse student course
```

Questo genererà la seguente migrazione:

```
class CreateJoinTableStudentCourse < ActiveRecord::Migration[5.0]
  def change
    create_join_table :students, :courses do |t|
      # t.index [:student_id, :course_id]
      # t.index [:course_id, :student_id]
    end
  end
end
```

## Esecuzione di migrazioni in diversi ambienti

Per eseguire le migrazioni nell'ambiente di `test` , eseguire questo comando shell:

```
rake db:migrate RAILS_ENV=test
```

5.0

A partire da Rails 5.0, puoi utilizzare le `rails` anziché il `rake` :

```
rails db:migrate RAILS_ENV=test
```

## Aggiungi una nuova colonna a una tabella

Per aggiungere un nuovo `name` colonna alla tabella `users` , eseguire il comando:

```
rails generate migration AddNameToUsers name
```

Questo genera la seguente migrazione:

```
class AddNameToUsers < ActiveRecord::Migration[5.0]
  def change
    add_column :users, :name, :string
  end
end
```

Quando il nome della migrazione è del formato `AddXXXToTABLE_NAME` seguito dall'elenco di colonne con tipi di dati, la migrazione generata conterrà le istruzioni `add_column` appropriate.

## Aggiungi una nuova colonna con un indice

Per aggiungere una nuova `email` *indicizzata alla* colonna degli `users` , eseguire il comando:

```
rails generate migration AddEmailToUsers email:string:index
```

Questo genererà la seguente migrazione:

```
class AddEmailToUsers < ActiveRecord::Migration[5.0]
  def change
    add_column :users, :email, :string
    add_index :users, :email
  end
end
```

## Rimuovi una colonna esistente da una tabella

Per rimuovere il `name` colonna esistente dalla tabella degli `users` , eseguire il comando:

```
rails generate migration RemoveNameFromUsers name:string
```

Questo genererà la seguente migrazione:

```
class RemoveNameFromUsers < ActiveRecord::Migration[5.0]
```

```
  def change
    remove_column :users, :name, :string
  end
end
```

Quando il nome della migrazione è nella forma `RemoveXXXFromYYY` seguito dall'elenco di colonne con tipi di dati, la migrazione generata conterrà le istruzioni `remove_column` appropriate.

Sebbene non sia necessario specificare il tipo di dati (ad esempio `:string` ) come parametro per `remove_column` , è altamente raccomandato. Se il tipo di dati *non* è specificato, la migrazione non sarà reversibile.

## Aggiungi una colonna di riferimento a una tabella

Per aggiungere un riferimento a un `team` alla tabella `users` , esegui questo comando:

```
$ rails generate migration AddTeamRefToUsers team:references
```

Questo genera la seguente migrazione:

```
class AddTeamRefToUsers < ActiveRecord::Migration[5.0]
  def change
    add_reference :users, :team, foreign_key: true
  end
end
```

Quella migrazione creerà una colonna `team_id` nella tabella degli `users` .

Se si desidera aggiungere un `index` appropriato e una `foreign_key` nella colonna aggiunta, modificare il comando in `rails generate migration AddTeamRefToUsers team:references:index` . Questo genererà la seguente migrazione:

```
class AddTeamRefToUsers < ActiveRecord::Migration
  def change
    add_reference :users, :team, index: true
    add_foreign_key :users, :teams
  end
end
```

Se si desidera denominare la colonna di riferimento diversa da quella che viene generata automaticamente da Rails, aggiungere quanto segue alla migrazione: (Ad esempio, è possibile chiamare l' `User` che ha creato il `Post` come `Author` nella tabella `Post` )

```
class AddAuthorRefToPosts < ActiveRecord::Migration
  def change
    add_reference :posts, :author, references: :users, index: true
  end
end
```

## Crea una nuova tabella

Per creare una nuova tabella `users` con il `name` e lo `salary` delle colonne, esegui il comando:

```
rails generate migration CreateUsers name:string salary:decimal
```

Questo genererà la seguente migrazione:

```
class CreateUsers < ActiveRecord::Migration[5.0]
  def change
    create_table :users do |t|
      t.string :name
      t.decimal :salary
    end
  end
end
```

Quando il nome della migrazione è del formato `CreateXXX` seguito dall'elenco di colonne con tipi di dati, verrà generata una migrazione che crea la tabella `XXX` con le colonne elencate.

## Aggiunta di più colonne a una tabella

Per aggiungere più colonne a una tabella, separare il `field:type` coppie con spazi quando si utilizzano i `rails generate migration` comando di `rails generate migration` .

La sintassi generale è:

```
rails generate migration NAME [field[:type][:index] field[:type][:index]] [options]
```

Ad esempio, quanto segue aggiungerà i campi `name` , `salary` ed `email` alla tabella `users` :

```
rails generate migration AddDetailsToUsers name:string salary:decimal email:string
```

Che genera la seguente migrazione:

```
class AddDetailsToUsers < ActiveRecord::Migration[5.0]
  def change
    add_column :users, :name, :string
    add_column :users, :salary, :decimal
    add_column :users, :email, :string
  end
end
```

## Esecuzione di migrazioni

Esegui comando:

5.0

```
rake db:migrate
```

5.0

```
rails db:migrate
```

Specificando la versione di destinazione verranno eseguite le migrazioni richieste (su, giù, cambia) fino a quando non avrà raggiunto la versione specificata. Qui, il `version number` è il prefisso numerico sul nome file della migrazione.

5.0

```
rake db:migrate VERSION=20080906120000
```

5.0

```
rails db:migrate VERSION=20080906120000
```

## Migrazioni di rollback

Per eseguire il `rollback` della migrazione più recente, ripristinando il metodo di `change` o eseguendo il metodo `down` . Esegui comando:

5.0

```
rake db:rollback
```

5.0

```
rails db:rollback
```

## Ripristina le ultime 3 migrazioni

5.0

```
rake db:rollback STEP=3
```

5.0

```
rails db:rollback STEP=3
```

`STEP` fornisce il numero di migrazioni da ripristinare.

## Rollback di tutte le migrazioni

5.0

```
rake db:rollback VERSION=0
```

5.0

```
rails db:rollback VERSION=0
```

## Cambiare le tabelle

Se hai creato una tabella con uno schema sbagliato, il modo più semplice per cambiare le colonne e le loro proprietà è `change_table` . Esamina il seguente esempio:

```
change_table :orders do |t|
  t.remove :ordered_at # removes column ordered_at
  t.string :skew_number # adds a new column
  t.index  :skew_number #creates an index
  t.rename :location, :state #renames location column to state
end
```

La precedente migrazione modifica gli `orders` una tabella. Ecco una descrizione riga per riga delle modifiche:

1. `t.remove :ordered_at` rimuove la colonna `ordered_at` dagli `orders` della tabella.
2. `t.string :skew_number` aggiunge una nuova colonna di tipo stringa denominata `skew_number` nella tabella degli `orders` .
3. `t.index :skew_number` aggiunge un indice sulla colonna `skew_number` nella tabella degli `orders` .
4. `t.rename :location, :state` rinomina la `location` colonna nella `orders` tabella `state` .

## Aggiungi una colonna univoca a una tabella

Per aggiungere una nuova `email` colonna *unica* agli `users` , eseguire il seguente comando:

```
rails generate migration AddEmailToUsers email:string:uniq
```

Questo creerà la seguente migrazione:

```
class AddEmailToUsers < ActiveRecord::Migration[5.0]
  def change
    add_column :users, :email, :string
    add_index :users, :email, unique: true
  end
end
```

## Cambia il tipo di una colonna esistente

Per modificare una colonna esistente in Rails con una migrazione, eseguire il seguente comando:

```
rails g migration change_column_in_table
```

Questo creerà un nuovo file di migrazione nella directory `db/migration` (se non esiste già), che conterrà il file con il prefisso timestamp e il nome del file di migrazione che contiene il contenuto seguente:

```
def change
  change_column(:table_name, :column_name, :new_type)
end
```

# Un metodo più lungo ma più sicuro

Il codice precedente impedisce all'utente di interrompere la migrazione. È possibile evitare questo problema suddividendo il metodo di `change` in metodi separati `up` e `down` :

```
def up
  change_column :my_table, :my_column, :new_type
end

def down
  change_column :my_table, :my_column, :old_type
end
```

## Ripeti le migrazioni

È possibile eseguire il rollback e quindi eseguire nuovamente la migrazione utilizzando il comando `redo` . Questo è fondamentalmente un collegamento che combina il `rollback` e `migrate` attività.

Esegui comando:

5.0

```
rake db:migrate:redo
```

5.0

```
rails db:migrate:redo
```

È possibile utilizzare il parametro `STEP` per tornare più di una versione.

Ad esempio, per tornare indietro 3 migrazioni:

5.0

```
rake db:migrate:redo STEP=3
```

5.0

```
rails db:migrate:redo STEP=3
```

## Aggiungi colonna con valore predefinito

L'esempio seguente aggiunge un `admin` colonna alla tabella `users` e assegna a tale colonna il valore predefinito `false` .

```
class AddDetailsToUsers < ActiveRecord::Migration[5.0]
  def change
    add_column :users, :admin, :boolean, default: false
```

```
  end
end
```

Le migrazioni con valori predefiniti potrebbero impiegare molto tempo in tabelle di grandi dimensioni, ad esempio PostgreSQL. Questo perché ogni riga dovrà essere aggiornata con il valore predefinito per la colonna appena aggiunta. Per aggirare questo problema (e ridurre i tempi di inattività durante le distribuzioni), puoi dividere la migrazione in tre passaggi:

1. Aggiungi un `add_column` -migration simile a quello sopra, ma non impostare alcun valore predefinito
2. Distribuisci e aggiorna la colonna in un'attività rake o sulla console mentre la tua app è in esecuzione. Assicurati che l'applicazione scriva già i dati su quella colonna per le righe nuove / aggiornate.
3. Aggiungi un'altra migrazione `change_column` , che quindi modifica il valore predefinito di tale colonna sul valore predefinito desiderato

## Vieta valori nulli

Per evitare valori `null` nelle colonne della tabella, aggiungi il parametro `:null` alla tua migrazione, in questo modo:

```
class AddPriceToProducts < ActiveRecord::Migration
  def change
    add_column :products, :float, null: false
  end
end
```

## Controllo dello stato della migrazione

Possiamo controllare lo stato delle migrazioni eseguendo

3.0 5.0

```
rake db:migrate:status
```

5.0

```
rails db:migrate:status
```

L'output sarà simile a questo:

```
Status   Migration ID    Migration Name
--------------------------------------------------
up     20140711185212  Create documentation pages
up     20140724111844  Create nifty attachments table
up     20140724114255  Create documentation screenshots
up     20160213170731  Create owners
up     20160218214551  Create users
up     20160221162159  ********** NO FILE **********
up     20160222231219  ********** NO FILE **********
```

Sotto il campo dello stato, `up` significa che la migrazione è stata eseguita e `down` significa che è necessario eseguire la migrazione.

### Crea una colonna di hstore

`Hstore` colonne di `Hstore` possono essere utili per memorizzare le impostazioni. Sono disponibili nei database PostgreSQL dopo aver abilitato l'estensione.

```
class CreatePages < ActiveRecord::Migration[5.0]
  def change
    create_table :pages do |t|
      enable_extension 'hstore' unless extension_enabled?('hstore')
      t.hstore :settings
      t.timestamps
    end
  end
end
```

### Aggiungi un riferimento personale

Un riferimento personale può essere utile per costruire un albero gerarchico. Questo può essere ottenuto con `add_reference` in una migrazione.

```
class AddParentPages < ActiveRecord::Migration[5.0]
  def change
    add_reference :pages, :pages
  end
end
```

La colonna chiave esterna sarà `pages_id` . Se si desidera decidere il nome della colonna chiave esterna, è necessario creare prima la colonna e aggiungere il riferimento dopo.

```
class AddParentPages < ActiveRecord::Migration[5.0]
  def change
    add_column :pages, :parent_id, :integer, null: true, index: true
    add_foreign_key :pages, :pages, column: :parent_id
  end
end
```

### Crea una colonna di array

Una colonna di `array` è supportata da PostgreSQL. Rails convertirà automaticamente un array PostgreSQL in un array Ruby e viceversa.

Crea una tabella con una colonna di `array` :

```
create_table :products do |t|
  t.string :name
  t.text :colors, array: true, default: []
end
```

Aggiungi una colonna di `array` a una tabella esistente:

```
add_column :products, :colors, array: true, default: []
```

Aggiungi un indice per una colonna di `array` :

```
add_index :products, :colors, using: 'gin'
```

## Aggiunta di un vincolo NOT NULL ai dati esistenti

Supponiamo che tu voglia aggiungere una chiave `company_id` alla tabella `users` e che tu debba avere un vincolo `NOT NULL` su di essa. Se hai già dati negli `users` , dovrai farlo in più passaggi.

```
class AddCompanyIdToUsers < ActiveRecord::Migration
  def up
    # add the column with NULL allowed
    add_column :users, :company_id, :integer

    # make sure every row has a value
    User.find_each do |user|
      # find the appropriate company record for the user
      # according to your business logic
      company = Company.first
      user.update!(company_id: company.id)
    end

    # add NOT NULL constraint
    change_column_null :users, :company_id, false
  end

  # Migrations that manipulate data must use up/down instead of change
  def down
    remove_column :users, :company_id
  end
end
```

Leggi Migrazioni di ActiveRecord online: https://riptutorial.com/it/ruby-on-rails/topic/1087/migrazioni-di-activerecord

# Capitolo 43: Modifica il fuso orario predefinito

## Osservazioni

> **config.active_record.default_timezone** determina se utilizzare Time.local (se impostato su: local) o Time.utc (se impostato su: utc) quando si estraggono date e ore dal database. L'impostazione predefinita è: utc.
> http://guides.rubyonrails.org/configuring.html

---

Se si desidera modificare il fuso orario **Rails** , ma continuare ad avere il **record attivo** salvato nel database in **UTC** , utilizzare

```
# application.rb
config.time_zone = 'Eastern Time (US & Canada)'
```

---

Se si desidera modificare il fuso orario **Rails e** avere tempi di archivio **Active Record** in questo fuso orario, utilizzare

```
# application.rb
config.time_zone = 'Eastern Time (US & Canada)'
config.active_record.default_timezone = :local
```

**Attenzione** : dovresti pensarci due volte, anche tre volte, prima di salvare i tempi nel database in un formato non UTC.

> **Nota**
> Non dimenticare di riavviare il server Rails dopo aver modificato `application.rb` .

---

Ricorda che `config.active_record.default_timezone` può richiedere solo due valori

- **: local** (converte nel fuso orario definito in `config.time_zone` )
- **: utc** (converte in UTC)

---

Ecco come puoi trovare tutti i fusi orari disponibili

```
rake time:zones:all
```

## Examples

### Cambia fuso orario Rails, ma continua ad avere il record attivo salvato nel

### database in UTC

```
# application.rb
config.time_zone = 'Eastern Time (US & Canada)'
```

### Cambia fuso orario Rails E hai orari di registrazione Active Record in questo fuso orario

```
# application.rb
config.time_zone = 'Eastern Time (US & Canada)'
config.active_record.default_timezone = :local
```

Leggi Modifica il fuso orario predefinito online: https://riptutorial.com/it/ruby-on-rails/topic/3367/modifica-il-fuso-orario-predefinito

# Capitolo 44: Modificare un ambiente dell'applicazione Rails predefinito

## introduzione

Questo discuterà su come cambiare l'ambiente in modo tale che quando qualcuno digita le `rails s` non si avviano nello sviluppo ma nell'ambiente che vogliono.

## Examples

### Funzionando su una macchina locale

Normalmente quando l'ambiente delle rotaie viene eseguito digitando. Questo esegue solo l'ambiente predefinito che di solito è lo `development`

```
rails s
```

L'ambiente specifico può essere selezionato usando il flag `-e` per esempio:

```
rails s -e test
```

Quale eseguirà l'ambiente di test.

L'ambiente predefinito può essere modificato nel terminale modificando il file `~/.bashrc` e aggiungendo la seguente riga:

```
export RAILS_ENV=production in your
```

### Funzionando su un server

Se si esegue su un server remoto che utilizza Passenger, modificare apache.conf nell'ambiente che si desidera utilizzare. Ad esempio in questo caso vedi la `RailsEnv production` .

```
<VirtualHost *:80>
  ServerName application_name.rails.local
  DocumentRoot "/Users/rails/application_name/public"
  RailsEnv production ## This is the default
</VirtualHost>
```

Leggi Modificare un ambiente dell'applicazione Rails predefinito online: https://riptutorial.com/it/ruby-on-rails/topic/9915/modificare-un-ambiente-dell-applicazione-rails-predefinito

# Capitolo 45: Modulo annidato in Ruby on Rails

## Examples

### Come configurare un modulo annidato in Ruby on Rails

La prima cosa da avere: un modello che contiene una relazione `has_many` con un altro modello.

```
class Project < ApplicationRecord
  has_many :todos
end

class Todo < ApplicationRecord
  belongs_to :project
end
```

In `ProjectsController` :

```
class ProjectsController < ApplicationController
  def new
    @project = Project.new
  end
end
```

In un modulo annidato, è possibile creare oggetti figlio con un oggetto genitore allo stesso tempo.

```
<%= nested_form_for @project do |f| %>
  <%= f.label :name %>
  <%= f.text_field :name %>

  <% # Now comes the part for `Todo` object %>
  <%= f.fields_for :todo do |todo_field| %>
    <%= todo_field.label :name %>
    <%= todo_field.text_field :name %>
  <% end %>
<% end %>
```

Come abbiamo inizializzato `@project` con `Project.new` per avere qualcosa per creare un nuovo oggetto `Project` , allo stesso modo per creare un oggetto `Todo` , dobbiamo avere qualcosa di simile, e ci sono diversi modi per farlo:

1. In `Projectscontroller` , in un `new` metodo, puoi scrivere: `@todo = @project.todos.build` o `@todo = @project.todos.new` per istanziare un nuovo oggetto `Todo` .
2. Puoi anche farlo in vista: `<%= f.fields_for :todos, @project.todos.build %>`

Per i parametri forti, puoi includerli nel modo seguente:

```
def project_params
  params.require(:project).permit(:name, todo_attributes: [:name])
end
```

Poiché, gli oggetti `Todo` verranno creati mediante la creazione di un oggetto `Project` , quindi devi specificare questa cosa nel modello `Project` aggiungendo la seguente riga:

```
accepts_nested_attributes_for :todos
```

Leggi Modulo annidato in Ruby on Rails online: https://riptutorial.com/it/ruby-on-rails/topic/8203/modulo-annidato-in-ruby-on-rails

# Capitolo 46: Mongoid

## Examples

### Installazione

Innanzitutto, aggiungi `Mongoid` al tuo `Gemfile` :

```
gem "mongoid", "~> 4.0.0"
```

e quindi eseguire l' `bundle install` . Oppure esegui semplicemente:

```
$ gem install mongoid
```

Dopo l'installazione, eseguire il generatore per creare il file di configurazione:

```
$ rails g mongoid:config
```

che creerà il file `(myapp)/config/mongoid.yml` .

### Creare un modello

Crea un modello (chiamiamolo `User` ) eseguendo:

```
$ rails g model User
```

che genererà il file `app/models/user.rb` :

```
class User
  include Mongoid::Document

end
```

Questo è tutto ciò che serve per avere un modello (anche se non un campo `id` ). A differenza di `ActiveRecord` , non ci sono file di migrazione. Tutte le informazioni del database per il modello sono contenute nel file del modello.

I timestamp non vengono automaticamente inclusi nel modello quando viene generato. Per aggiungere `created_at` e `updated_at` al tuo modello, aggiungi

```
include Mongoid::Timestamps
```

al tuo modello di sotto `include Mongoid::Document` come questo:

```
class User
  include Mongoid::Document
```

```
  include Mongoid::Timestamps

end
```

## campi

Come da documentazione Mongoid , ci sono 16 tipi di campi validi:

- schieramento
- BigDecimal
- booleano
- Data
- Appuntamento
- Galleggiante
- hash
- Numero intero
- BSON :: ObjectId
- BSON :: binario
- Gamma
- regexp
- Stringa
- Simbolo
- Tempo
- TimeWithZone

Per aggiungere un campo (chiamiamolo `name` e lo facciamo essere una `String` ), aggiungi questo al tuo file di modello:

```
field :name, type: String
```

Per impostare un valore predefinito, basta passare l'opzione `default` :

```
field :name, type: String, default: ""
```

## Associazioni classiche

Mongoid consente le classiche associazioni `ActiveRecord` :

- Uno a uno: `has_one` / `belongs_to`
- Uno a molti: `has_many` / `belongs_to`
- Molti-a-molti: `has_and_belongs_to_many`

Per aggiungere un'associazione (diciamo che l'utente ha `has_many` messaggi), puoi aggiungerla al tuo file modello `User` :

```
has_many :posts
```

e questo al tuo file modello `Post` :

```
belongs_to :user
```

Questo aggiungerà un campo `user_id` nel tuo modello `Post` , aggiungerà un metodo `user` alla tua classe `Post` e aggiungerà un metodo `posts` alla tua classe `User` .

## Associazioni incorporate

Mongoid consente le associazioni incorporate:

- Uno a uno: `embeds_one` / `embedded_in`
- Uno-a-molti: `embeds_many` / `embedded_in`

Per aggiungere un'associazione (diciamo che l'utente `embeds_many` indirizzi), aggiungi questo al tuo file `User` :

```
embeds_many :addresses
```

e questo al tuo file modello `Address` :

```
embedded_in :user
```

Questo incorpora l' `Address` nel tuo modello `User` , aggiungendo un metodo di `addresses` alla tua classe `User` .

## Chiamate di database

Mongoid prova ad avere una sintassi simile a `ActiveRecord` quando può. Supporta queste chiamate (e molte altre)

```
User.first #Gets first user from the database

User.count #Gets the count of all users from the database

User.find(params[:id]) #Returns the user with the id found in params[:id]

User.where(name: "Bob") #Returns a Mongoid::Criteria object that can be chained
                        #with other queries (like another 'where' or an 'any_in')
                        #Does NOT return any objects from database

User.where(name: "Bob").entries #Returns all objects with name "Bob" from database

User.where(:name.in => ['Bob', 'Alice']).entries #Returns all objects with name "Bob" or
"Alice" from database

User.any_in(name: ["Bob", "Joe"]).first #Returns the first object with name "Bob" or "Joe"
User.where(:name => 'Bob').exists? # will return true if there is one or more users with name
bob
```

Leggi Mongoid online: https://riptutorial.com/it/ruby-on-rails/topic/3071/mongoid

# Capitolo 47: Motivo decorativo

## Osservazioni

Il **pattern Decorator** consente di aggiungere o modificare il comportamento degli oggetti in modo situazionale senza influire sull'oggetto di base.

Questo può essere ottenuto attraverso il semplice Ruby usando lo stdlib, o tramite gemme popolari come Draper .

## Examples

### Decorazione di un modello con SimpleDelegator

La maggior parte degli sviluppatori di Rails inizia modificando le informazioni sul modello all'interno del modello stesso:

```
<h1><%= "#{ @user.first_name } #{ @user.last_name }" %></h1>
<h3>joined: <%= @user.created_at.in_time_zone(current_user.timezone).strftime("%A, %d %b %Y
%l:%M %p") %></h3>
```

Per i modelli con molti dati, questo può diventare rapidamente ingombrante e portare alla logica di copia-incolla da un modello all'altro.

Questo esempio utilizza `SimpleDelegator` dallo stdlib.

Tutte le richieste a un oggetto `SimpleDelegator` vengono passate all'oggetto padre per impostazione predefinita. È possibile sovrascrivere qualsiasi metodo con la logica di presentazione oppure è possibile aggiungere nuovi metodi specifici per questa vista.

`SimpleDelegator` fornisce due metodi: `__setobj__` per impostare a quale oggetto viene delegato e `__getobj__` per ottenere quell'oggetto.

```
class UserDecorator < SimpleDelegator
  attr_reader :view
  def initialize(user, view)
    __setobj__ @user
    @view = view
  end

  # new methods can call methods on the parent implicitly
  def full_name
    "#{ first_name } #{ last_name }"
  end

  # however, if you're overriding an existing method you need
  # to use __getobj__
  def created_at
    Time.use_zone(view.current_user.timezone) do
```

```
      __getobj__.created_at.strftime("%A, %d %b %Y %l:%M %p")
    end
  end
end
```

Alcuni decoratori si affidano alla magia per collegare questo comportamento, ma è possibile rendere più ovvio da dove proviene la logica di presentazione inizializzando l'oggetto sulla pagina.

```
<% user = UserDecorator.new(@user, self) %>
<h1><%= user.full_name %></h1>
<h3>joined: <%= user.created_at %></h3>
```

Passando un riferimento all'oggetto vista nel decoratore, possiamo ancora accedere a tutti gli altri helper della vista mentre si costruisce la logica di presentazione senza doverlo includere.

Ora il modello di visualizzazione riguarda solo l'inserimento di dati nella pagina ed è molto più chiaro.

## Decorare un modello usando Draper

Draper abbina automaticamente i modelli con i loro decoratori per convenzione.

```
# app/decorators/user_decorator.rb
class UserDecorator < Draper::Decorator
  def full_name
    "#{object.first_name} #{object.last_name}"
  end

  def created_at
    Time.use_zone(h.current_user.timezone) do
      object.created_at.strftime("%A, %d %b %Y %l:%M %p")
    end
  end
end
```

Data una variabile `@user` che contiene un oggetto ActiveRecord, puoi accedere al decoratore chiamando `#decorate` su `@user` o specificando la classe Draper se vuoi essere specifico.

```
<% user = @user.decorate %><!-- OR -->
<% user = UserDecorator.decorate(@user) %>
<h1><%= user.full_name %></h1>
<h3>joined: <%= user.created_at %></h3>
```

Leggi Motivo decorativo online: https://riptutorial.com/it/ruby-on-rails/topic/5694/motivo-decorativo

# Capitolo 48: Operaia

## Examples

### Definire le fabbriche

Se si dispone di una classe utente ActiveRecord con attributi di nome ed e-mail, è possibile creare una fabbrica facendo in modo che FactoryGirl lo indovini:

```
FactoryGirl.define do
  factory :user do # it will guess the User class
    name     "John"
    email    "john@example.com"
  end
end
```

Oppure puoi renderlo esplicito e persino cambiarne il nome:

```
FactoryGirl.define do
  factory :user_jack, class: User do
    name     "Jack"
    email    "jack@example.com"
  end
end
```

Poi nelle tue specifiche puoi usare i metodi di FactoryGirl con questi, in questo modo:

```
# To create a non saved instance of the User class filled with John's data
build(:user)
# and to create a non saved instance of the User class filled with Jack's data
build(:user_jack)
```

I metodi più comuni sono:

```
# Build returns a non saved instance
user = build(:user)

# Create returns a saved instance
user = create(:user)

# Attributes_for returns a hash of the attributes used to build an instance
attrs = attributes_for(:user)
```

Leggi Operaia online: https://riptutorial.com/it/ruby-on-rails/topic/8330/operaia

# Capitolo 49: Organizzazione di classe

## Osservazioni

Sembra una cosa semplice da fare, ma quando le lezioni inizieranno a gonfiarsi, sarai grato di aver trovato il tempo per organizzarle.

## Examples

### Classe di modello

```
class Post < ActiveRecord::Base
  belongs_to :user
  has_many :comments

  validates :user, presence: true
  validates :title, presence: true, length: { in: 6..40 }

  scope :topic, -> (topic) { joins(:topics).where(topic: topic) }

  before_save :update_slug
  after_create :send_welcome_email

  def publish!
    update(published_at: Time.now, published: true)
  end

  def self.find_by_slug(slug)
    find_by(slug: slug)
  end

  private

  def update_slug
    self.slug = title.join('-')
  end

  def send_welcome_email
    WelcomeMailer.welcome(self).deliver_now
  end
end
```

I modelli sono in genere responsabili di:

- instaurare relazioni
- dati di convalida
- fornendo accesso ai dati tramite ambiti e metodi
- Esecuzione di azioni sulla persistenza dei dati.

Al livello più alto, i modelli descrivono concetti di dominio e ne gestiscono la persistenza.

## Classe di servizio

Il controller è un punto di accesso alla nostra applicazione. Tuttavia, non è l'unico punto di ingresso possibile. Mi piacerebbe avere la mia logica accessibile da:

- Rake tasks
- lavori in background
- consolle
- test

Se lancio la mia logica in un controller, non sarà accessibile da tutti questi posti. Quindi proviamo con l'approccio "skinny controller, fat model" e spostiamo la logica su un modello. Ma quale? Se un dato pezzo di logica coinvolge modelli `User` , `Cart` e `Product` - dove dovrebbe vivere?

Una classe che eredita da `ActiveRecord::Base` ha già molte responsabilità. Gestisce l'interfaccia di query, le associazioni e le convalide. Se aggiungi ancora più codice al tuo modello, diventerà rapidamente un disordine irraggiungibile con centinaia di metodi pubblici.

Un servizio è solo un normale oggetto Ruby. La sua classe non deve ereditare da una classe specifica. Il suo nome è una frase verbale, ad esempio `CreateUserAccount` anziché `UserCreation` o `UserCreationService` . Vive nella directory app / servizi. Devi creare questa directory da solo, ma Rails caricherà automaticamente le lezioni per te.

**Un oggetto di servizio fa una cosa**

Un oggetto servizio (noto anche come oggetto metodo) esegue un'azione. Tiene la logica aziendale per eseguire quell'azione. Ecco un esempio:

```
# app/services/accept_invite.rb
class AcceptInvite
  def self.call(invite, user)
    invite.accept!(user)
    UserMailer.invite_accepted(invite).deliver
  end
end
```

Le tre convenzioni che seguo sono:

I servizi vanno nella `app/services directory` . Ti incoraggio a utilizzare le sottodirectory per domini di logica aziendale-pesante. Per esempio:

- Il file `app/services/invite/accept.rb` definirà `Invite::Accept` mentre `app/services/invite/create.rb` definirà `Invite::Create`
- I servizi iniziano con un verbo (e non terminano con il servizio): `ApproveTransaction` , `SendTestNewsletter` , `ImportUsersFromCsv`
- I servizi rispondono al metodo di `call` . Ho trovato che usare un altro verbo lo rende un po 'ridondante: `ApproveTransaction.approve()` non legge bene. Inoltre, il metodo `call` è il metodo de facto per `lambda` , `procs` e gli oggetti metodo.

**Benefici**

***Gli oggetti di servizio mostrano cosa fa la mia applicazione***

Posso semplicemente dare un'occhiata alla directory dei servizi per vedere cosa fa la mia applicazione: `ApproveTransaction` , `CancelTransaction` , `BlockAccount` , `SendTransactionApprovalReminder` ...

Un rapido sguardo su un oggetto di servizio e so quale logica aziendale è coinvolta. Non devo passare attraverso i controller, i callback e gli osservatori del modello `ActiveRecord` per capire cosa implica "approvare una transazione".

***Modelli e controller di pulizia***

I controllori trasformano la richiesta (parametri, sessione, cookie) in argomenti, li inoltrano al servizio e reindirizzano o eseguono il rendering in base alla risposta del servizio.

```
class InviteController < ApplicationController
 def accept
    invite = Invite.find_by_token!(params[:token])
    if AcceptInvite.call(invite, current_user)
      redirect_to invite.item, notice: "Welcome!"
    else
      redirect_to '/', alert: "Oopsy!"
    end
  end
end
```

I modelli si occupano solo di associazioni, scopi, convalide e persistenza.

```
class Invite < ActiveRecord::Base
  def accept!(user, time=Time.now)
    update_attributes!(
      accepted_by_user_id: user.id,
      accepted_at: time
    )
  end
end
```

Questo rende i modelli e i controller molto più facili da testare e mantenere!

**Quando utilizzare la classe di servizio**

Raggiungi gli oggetti di servizio quando un'azione risponde a uno o più di questi criteri:

- L'azione è complessa (ad esempio chiudendo i libri alla fine di un periodo contabile)
- L'azione raggiunge più modelli (ad esempio un acquisto di e-commerce utilizzando oggetti Order, CreditCard e Customer)
- L'azione interagisce con un servizio esterno (ad es. Postare sui social network)
- L'azione non è una preoccupazione centrale del modello sottostante (ad es. Spazzare dati obsoleti dopo un certo periodo di tempo).
- Esistono diversi modi per eseguire l'azione (ad esempio l'autenticazione con un token di accesso o una password).

**fonti**

Adam Niedzielski Blog

Brew House Blog

Codice clima blog

Leggi Organizzazione di classe online: https://riptutorial.com/it/ruby-on-rails/topic/7623/organizzazione-di-classe

# Capitolo 50: Parole riservate

## introduzione

Dovresti stare attento a usare queste parole per variabili, nome del modello, nome del metodo o ecc.

## Examples

### Elenco delle parole riservate

- ADDITIONAL_LOAD_PATHS
- ARGF
- ARGV
- ActionController
- ActionView
- ActiveRecord
- ArgumentError
- schieramento
- BasicSocket
- segno di riferimento
- bignum
- Rilegatura
- CGI
- CGIMethods
- CROSS_COMPILING
- Classe
- ClassInheritableAttributes
- paragonabile
- ConditionVariable
- config
- continuazione
- DRB
- DRbIdConv
- DRbObject
- DRbUndumped
- Dati
- Data
- Appuntamento
- Delegater
- delegante
- digerire
- dir
- ENV

- EOFError
- ERB
- Enumerable
- errno
- Eccezione
- FALSE
- FalseClass
- fcntl
- File
- FileList
- FileTask
- filestat
- fileutils
- Fixnum
- Galleggiante
- FloatDomainError
- GC
- gemma
- GetoptLong
- hash
- IO
- IOError
- IPSocket
- IPsocket
- IndexError
- inflettore
- Numero intero
- Interrompere
- nocciolo
- LN_SUPPORTED
- LoadError
- LocalJumpError
- logger
- Maresciallo
- MatchData
- MatchingData
- Matematica
- Metodo
- Modulo
- mutex
- mysql
- MysqlError
- MysqlField
- MysqlRes
- NIL
- NameError

- NilClass
- NoMemoryError
- NoMethodError
- nowrite
- NotImplementedError
- Numerico
- OPT_TABLE
- Oggetto
- ObjectSpace
- Osservabile
- Osservatore
- PGError
- PGconn
- PGlarge
- PGresult
- PIATTAFORMA
- PStore
- parsedate
- Precisione
- proc
- Processi
- Coda
- RAKEVERSION
- DATA DI RILASCIO
- RUBINO
- RUBY_PLATFORM
- RUBY_RELEASE_DATE
- RUBY_VERSION
- cremagliera
- Rastrello
- RakeApp
- RakeFileUtils
- Gamma
- RangeError
- Razionale
- regexp
- RegexpError
- Richiesta
- RuntimeError
- STDERR
- STDIN
- STDOUT
- ScanError
- ScriptError
- Errore di sicurezza
- Segnale

- SignalException
- SimpleDelegater
- SimpleDelegator
- Singleton
- SizedQueue
- presa di corrente
- Errore socket
- Errore standard
- Stringa
- StringScanner
- struct
- Simbolo
- Errore di sintassi
- SystemCallError
- SystemExit
- SystemStackError
- TCPServer
- TCPSocket
- tcpserver
- TCPSocket
- TOPLEVEL_BINDING
- VERO
- Compito
- Testo
- Filo
- ThreadError
- ThreadGroup
- Tempo
- Transazione
- TrueClass
- TypeError
- UDPSocket
- UDPsocket
- UNIXServer
- UNIXSocket
- UNIXserver
- UNIXsocket
- UnboundMethod
- url
- VERSIONE
- verboso
- YAML
- ZeroDivisionError
- @base_path
- accettare
- Accesso

- Axi
- azione
- attributi
- Application2
- richiama
- categoria
- connessione
- Banca dati
- dispatcher
- display1
- guidare
- errori
- formato
- ospite
- chiave
- disposizione
- caricare
- collegamento
- nuovo
- notificare
- Aperto
- pubblico
- citazione
- rendere
- richiesta
- record
- risposte
- salvare
- scopo
- inviare
- sessione
- sistema
- modello
- test
- tempo scaduto
- to_s
- genere
- URI
- visite
- Osservatore

**Nomi dei campi del database**

- created_at
- creato
- updated_at
- updated_on

- deleted_at
- (paranoia
- gemma)
- lock_version
- genere
- id
- # {Nome_tabella} _count
- posizione
- parent_id
- LFT
- rgt
- quote_value

**Ruby Reserved Words**

- alias
- e
- INIZIO
- inizio
- rompere
- Astuccio
- classe
- DEF
- definito?
- fare
- altro
- ELSIF
- FINE
- fine
- garantire
- falso
- per
- Se
- modulo
- Il prossimo
- zero
- non
- o
- rifare
- salvare
- riprovare
- ritorno
- se stesso
- super
- poi
- vero
- undef

- salvo che
- fino a
- quando
- mentre
- dare la precedenza
- _ *FILE* _
- _ *LINE* _

Leggi Parole riservate online: https://riptutorial.com/it/ruby-on-rails/topic/10818/parole-riservate

# Capitolo 51: PDF di gamberi

## Examples

### Esempio avanzato

Questo è l'approccio avanzato con l'esempio

```
class FundsController < ApplicationController

  def index
    @funds = Fund.all_funds(current_user)
  end

  def show
    @fund = Fund.find(params[:id])
    respond_to do |format|
      format.html
      format.pdf do
        pdf = FundsPdf.new(@fund, view_context)
        send_data pdf.render, filename:
        "fund_#{@fund.created_at.strftime("%d/%m/%Y")}.pdf",
        type: "application/pdf"
      end
    end
  end
end
```

Io sopra il codice abbiamo questa linea `FundsPdf.new(@fund, view_context)` . Qui stiamo inizializzando la classe FundsPdf con @fund instance e view_context per usare i metodi helper in FundsPdf. FundsPdf sarebbe simile a questo

```
 class FundPdf < Prawn::Document

  def initialize(fund, view)
    super()
    @fund = fund
    @view = view
    upper_half
    lower_half
  end

  def upper_half
    logopath =  "#{Rails.root}/app/assets/images/logo.png"
    image logopath, :width => 197, :height => 91
    move_down 10
    draw_text "Receipt", :at => [220, 575], size: 22
    move_down 80
    text "Hello #{@invoice.customer.profile.first_name.capitalize},"
  end

  def thanks_message
    move_down 15
    text "Thank you for your order.Print this receipt as
```

```
    confirmation of your order.",
    :indent_paragraphs => 40, :size => 13
  end
end
```

Questo è uno dei migliori approcci per generare PDF con classi che utilizzano la gemma Prawn.

### Esempio di base

È necessario aggiungere Gem e PDF MIME: digitare all'interno di mime_types.rb in quanto è necessario notificare le rotaie sul tipo di mime PDF.

Dopodiché possiamo generare Pdf con Prawn seguendo i metodi di base

## Questo è l'incarico base

```
pdf = Prawn::Document.new
pdf.text "Hello World"
pdf.render_file "assignment.pdf"
```

## Possiamo farlo con Implicit Block

```
Prawn::Document.generate("implicit.pdf") do
 text "Hello World"
end
```

## Con il blocco esplicito

```
Prawn::Document.generate("explicit.pdf") do |pdf|
 pdf.text "Hello World"
end
```

Leggi PDF di gamberi online: https://riptutorial.com/it/ruby-on-rails/topic/4163/pdf-di-gamberi

# Capitolo 52: Percorso superficiale

## Examples

### 1. Uso di poco profondo

Un modo per evitare l'annidamento profondo (come raccomandato sopra) è generare le azioni di raccolta nell'ambito del genitore, in modo da avere un senso della gerarchia, ma non annidare le azioni dei membri. In altre parole, per costruire solo percorsi con la quantità minima di informazioni per identificare in modo univoco la risorsa, in questo modo:

```
resources :articles, shallow: true do
  resources :comments
  resources :quotes
  resources :drafts
end
```

Il metodo superficiale del DSL crea un ambito all'interno del quale ogni annidamento è superficiale. Questo genera gli stessi percorsi dell'esempio precedente:

```
shallow do
  resources :articles do
    resources :comments
    resources :quotes
    resources :drafts
  end
end
```

Esistono due opzioni per l'ambito di personalizzare i percorsi poco profondi. : shallow_path prefixa i percorsi dei membri con il parametro specificato:

```
scope shallow_path: "sekret" do
  resources :articles do
    resources :comments, shallow: true
  end
end
```

Usa il comando Rake per ottenere percorsi generati come definito di seguito:

```
rake routes
```

Leggi Percorso superficiale online: https://riptutorial.com/it/ruby-on-rails/topic/7775/percorso-superficiale

# Capitolo 53: Quadri rotaie nel corso degli anni

## introduzione

Quando sei nuovo su Rails e lavori sulle applicazioni legacy Rails, può essere difficile capire quale framework è stato introdotto quando. Questo argomento è progettato per essere l'elenco *definitivo* di tutti i framework attraverso le versioni di Rails.

## Examples

### Come trovare i framework disponibili nella versione corrente di Rails?

Utilizzare il

```
config.frameworks
```

opzione per ottenere una serie di `Symbol` che rappresentano ciascun quadro.

### Versioni di rails in Rails 1.x

- ActionMailer
- ActionPack
- ActionWebService
- ActiveRecord
- ActiveSupport
- Railties

### Strutture di rotaie in Rails 2.x

- ActionMailer
- ActionPack
- ActiveRecord
- ActiveResource ( *ActiveWebService è stato sostituito da ActiveResource e con questo Rails è passato da SOAP a REST per impostazione predefinita* )
- ActiveSupport
- Railties

### Strutture per rotaie in Rails 3.x

- ActionMailer
- ActionPack
- ActiveModel
- ActiveRecord

- ActiveResource
- ActiveSupport
- Railties

Leggi Quadri rotaie nel corso degli anni online: https://riptutorial.com/it/ruby-on-rails/topic/8107/quadri-rotaie-nel-corso-degli-anni

# Capitolo 54: Rails 5

## Examples

### Creazione di un'API Ruby on Rails 5

Per creare una nuova API Rails 5, apri un terminale ed esegui il seguente comando:

```
rails new app_name --api
```

Verrà creata la seguente struttura di file:

```
  create
  create  README.rdoc
  create  Rakefile
  create  config.ru
  create  .gitignore
  create  Gemfile
  create  app
  create  app/assets/javascripts/application.js
  create  app/assets/stylesheets/application.css
  create  app/controllers/application_controller.rb
  create  app/helpers/application_helper.rb
  create  app/views/layouts/application.html.erb
  create  app/assets/images/.keep
  create  app/mailers/.keep
  create  app/models/.keep
  create  app/controllers/concerns/.keep
  create  app/models/concerns/.keep
  create  bin
  create  bin/bundle
  create  bin/rails
  create  bin/rake
  create  bin/setup
  create  config
  create  config/routes.rb
  create  config/application.rb
  create  config/environment.rb
  create  config/secrets.yml
  create  config/environments
  create  config/environments/development.rb
  create  config/environments/production.rb
  create  config/environments/test.rb
  create  config/initializers
  create  config/initializers/assets.rb
  create  config/initializers/backtrace_silencers.rb
  create  config/initializers/cookies_serializer.rb
  create  config/initializers/filter_parameter_logging.rb
  create  config/initializers/inflections.rb
  create  config/initializers/mime_types.rb
  create  config/initializers/session_store.rb
  create  config/initializers/wrap_parameters.rb
  create  config/locales
  create  config/locales/en.yml
  create  config/boot.rb
```

```
  create  config/database.yml
  create  db
  create  db/seeds.rb
  create  lib
  create  lib/tasks
  create  lib/tasks/.keep
  create  lib/assets
  create  lib/assets/.keep
  create  log
  create  log/.keep
  create  public
  create  public/404.html
  create  public/422.html
  create  public/500.html
  create  public/favicon.ico
  create  public/robots.txt
  create  test/fixtures
  create  test/fixtures/.keep
  create  test/controllers
  create  test/controllers/.keep
  create  test/mailers
  create  test/mailers/.keep
  create  test/models
  create  test/models/.keep
  create  test/helpers
  create  test/helpers/.keep
  create  test/integration
  create  test/integration/.keep
  create  test/test_helper.rb
  create  tmp/cache
  create  tmp/cache/assets
  create  vendor/assets/javascripts
  create  vendor/assets/javascripts/.keep
  create  vendor/assets/stylesheets
  create  vendor/assets/stylesheets/.keep
```

Questa struttura di file verrà creata all'interno di una nuova cartella denominata `app_name` .
Contiene tutte le risorse e il codice necessari per avviare il tuo progetto.

Inserisci la cartella e installa le dipendenze:

```
cd app_name
bundle install
```

Dovresti anche iniziare il tuo database. Rails utilizza SQLite come database predefinito. Per crearlo, esegui:

```
rake db:setup
```

Ora esegui la tua appplicazione:

```
$ rails server
```

Quando apri il tuo browser su `http://localhost:3000` , la tua nuova (vuota) API dovrebbe essere in esecuzione!

## Come installare Ruby on Rails 5 su RVM

RVM è un ottimo strumento per gestire le tue versioni di ruby e impostare il tuo ambiente di lavoro.

Supponendo che tu abbia già installato RVM, per ottenere l'ultima versione di ruby, che è necessaria per questi esempi, apri un terminale ed esegui:

```
$ rvm get stable
$ rvm install ruby --latest
```

Controlla la tua versione ruby eseguendo:

```
$ ruby -v
> ruby 2.3.0p0
```

Per installare Rails 5, devi prima creare un nuovo gemset usando la versione più recente di Ruby e quindi installare i binari:

```
$ rvm use ruby-2.3.0@my_app --create
$ gem install rails
```

Per verificare la versione del tuo binario, esegui:

```
$ rails -v
> Rails 5.0.0
```

Leggi Rails 5 online: https://riptutorial.com/it/ruby-on-rails/topic/3019/rails-5

# Capitolo 55: Rails Cookbook - Advanced Rails: ricette / apprendimento e tecniche di codifica

## Examples

### Giocare con le tabelle usando la console di rails

---

**Visualizza tabelle**

```
ActiveRecord::Base.connection.tables
```

**Elimina qualsiasi tabella** .

```
    ActiveRecord::Base.connection.drop_table("users")
    ------------OR----------------------
    ActiveRecord::Migration.drop_table(:users)
    ------------OR----------------------
    ActiveRecord::Base.connection.execute("drop table users")
```

**Rimuovi l'indice dalla colonna esistente**

```
 ActiveRecord::Migration.remove_index(:users, :name => 'index_users_on_country')
```

dove `country` è un nome di colonna nel file di migrazione con indice **già** aggiunto nella tabella `users` come mostrato di seguito: -

```
 t.string :country,add_index: true
```

**Rimuovi il vincolo della chiave esterna**

```
ActiveRecord::Base.connection.remove_foreign_key('food_items', 'menus')
```

dove `menus has_many food_items` e le loro rispettive migrazioni.

**Aggiungi colonna**

```
ActiveRecord::Migration.remove_column :table_name, :column_name
```

per esempio:-

```
 ActiveRecord::Migration.add_column :profiles, :profile_likes, :integer, :default => 0
```

## Metodi di Rails - Restituzione di valori booleani

Qualsiasi metodo nel modello Rails può restituire il valore booleano.

**metodo semplice-**

```
##this method return ActiveRecord::Relation
def check_if_user_profile_is_complete
  User.includes( :profile_pictures,:address,:contact_detail).where("user.id = ?",self)
end
```

**Ancora un metodo semplice che restituisce il valore booleano-**

```
##this method return Boolean(NOTE THE !! signs before result)
def check_if_user_profile_is_complete
  !!User.includes( :profile_pictures,:address,:contact_detail).where("user.id = ?",self)
end
```

Quindi, lo stesso metodo ora restituirà booleano invece di qualsiasi altra cosa :).

## Gestione dell'errore - metodo non definito `dove 'per #

A volte vogliamo utilizzare una query `where` su una raccolta di record restituita che non sia `ActiveRecord::Relation` Quindi otteniamo l'errore sopra riportato come clausola `Where` è nota a `ActiveRecord` e non a `Array` .

C'è una soluzione precisa per questo usando `Joins` .

**ESEMPIO** : -

> Supponiamo di dover trovare tutti i profili utente (UserProfile) che sono attivi e che non è un utente (utente) con un id = 10.

```
UserProfiles.includes(:user=>:profile_pictures]).where(:active=>true).map(&:user).where.not(:id=>10)
```

Così sopra interrogazione fallirà dopo `map` come `map` restituisce un `array` che **non** funziona con `where` clausola.

**Ma usando i join, lo farà funzionare,**

```
UserProfiles.includes(:user=>:profile_pictures]).where(:active=>true).joins(:user).where.not(:id=>10)
```

Poiché i `joins` genereranno record simili come `map` ma saranno `ActiveRecord` e **non** una `Array` .

Leggi Rails Cookbook - Advanced Rails: ricette / apprendimento e tecniche di codifica online: https://riptutorial.com/it/ruby-on-rails/topic/7259/rails-cookbook---advanced-rails--ricette---apprendimento-e-tecniche-di-codifica

# Capitolo 56: Rails Engine - Modular Rails

## introduzione

**Panoramica rapida dei motori Rails**

I motori sono piccole applicazioni Rails che possono essere utilizzate per aggiungere funzionalità all'applicazione che li ospita. La classe che definisce un'applicazione Ruby on Rails è `Rails::Application` che in realtà eredita gran parte del suo comportamento da `Rails::Engine` , la classe che definisce un motore. Possiamo dire che un'applicazione Rails regolare è semplicemente un motore con più funzionalità.

## Sintassi

- plugin per rails new my_module --mountable

## Examples

### Crea un'app modulare

# Iniziare

Innanzitutto, creiamo una nuova applicazione Ruby on Rails:

```
rails new ModularTodo
```

Il prossimo passo è generare un motore!

```
cd ModularTodo && rails plugin new todo --mountable
```

Creeremo anche una cartella 'motori' per immagazzinare i motori (anche se ne abbiamo uno solo!).

```
mkdir engines && mv todo ./engines
```

I motori, proprio come le gemme, sono dotati di un file gemspec. Mettiamo alcuni valori reali per evitare gli avvertimenti.

```
 #ModularTodo/engines/todo/todo.gemspec
$:.push File.expand_path("../lib", __FILE__)

#Maintain your gem's version:
require "todo/version"
```

```
#Describe your gem and declare its dependencies:
Gem::Specification.new do |s|
  s.name        = "todo"
  s.version     = Todo::VERSION
  s.authors     = ["Thibault Denizet"]
  s.email       = ["bo@samurails.com"]
  s.homepage    = "//samurails.com"
  s.summary     = "Todo Module"
  s.description = "Todo Module for Modular Rails article"
  s.license     = "MIT"

  #Moar stuff
  #...
end
```

Ora dobbiamo aggiungere il motore Todo all'applicazione madre Gemfile.

```
#ModularTodo/Gemfile
#Other gems
gem 'todo', path: 'engines/todo'
```

Eseguiamo l' `bundle install` . Dovresti vedere quanto segue nella lista delle gemme:

```
Using todo 0.0.1 from source at engines/todo
```

Ottimo, il nostro motore Todo è stato caricato correttamente! Prima di iniziare la codifica, abbiamo un'ultima cosa da fare: montare il motore Todo. Possiamo farlo nel file routes.rb nell'app padre.

```
Rails.application.routes.draw do
  mount Todo::Engine => "/", as: 'todo'
end
```

Lo stiamo montando su `/` ma potremmo renderlo accessibile anche in `/todo` . Dato che abbiamo un solo modulo, `/` sta bene.

Ora puoi avviare il tuo server e controllarlo nel tuo browser. Dovresti vedere la vista Rails di default perché non abbiamo ancora definito alcun controller / vista. Facciamolo adesso!

# Costruire la lista di cose da fare

Stiamo andando a impalcare un modello chiamato `Task` all'interno del modulo Todo ma per migrare correttamente il database dall'applicazione madre, abbiamo bisogno di aggiungere un piccolo inizializzatore al file `engine.rb` .

```
#ModularTodo/engines/todo/lib/todo/engine.rb
module Todo
  class Engine < ::Rails::Engine
    isolate_namespace Todo
```

```
    initializer :append_migrations do |app|
      unless app.root.to_s.match(root.to_s)
        config.paths["db/migrate"].expanded.each do |p|
          app.config.paths["db/migrate"] << p
        end
      end
    end

  end
end
```

È così, ora quando eseguiamo le migrazioni dall'applicazione principale, verranno caricate anche le migrazioni nel motore di Todo.

Creiamo il modello `Task` . Il comando `scaffold` deve essere eseguito dalla cartella del motore.

```
cd engines/todo && rails g scaffold Task title:string content:text
```

Esegui le migrazioni dalla cartella principale:

```
rake db:migrate
```

Ora, abbiamo solo bisogno di definire la root route all'interno del motore Todo:

```
#ModularTodo/engines/todo/config/routes.rb
Todo::Engine.routes.draw do
  resources :tasks
  root 'tasks#index'
end
```

Puoi giocarci, creare compiti, cancellarli ... Oh aspetta, l'eliminazione non funziona! Perché?! Bene, sembra che JQuery non sia caricato, quindi aggiungiamolo al file `application.js` all'interno del motore!

```
// ModularTodo/engines/todo/app/assets/javascripts/todo/application.js
//= require jquery
//= require jquery_ujs
//= require_tree .
```

Sì, ora possiamo distruggere i compiti!

Leggi Rails Engine - Modular Rails online: https://riptutorial.com/it/ruby-on-rails/topic/9080/rails-engine---modular-rails

# Capitolo 57: Rails -Engines

## introduzione

I motori possono essere considerati applicazioni in miniatura che forniscono funzionalità alle loro applicazioni host. Un'applicazione Rails è in realtà solo un motore "sovralimentato", con la classe Rails :: Application che eredita gran parte del suo comportamento da Rails :: Engine.

I motori sono le applicazioni / plug-in delle guide riutilizzabili. Funziona come una gemma. I motori famosi sono i gemelli Device, Spree che possono essere facilmente integrati con le applicazioni su rotaia.

## Sintassi

- `rails plugin new [engine name] --mountable`

## Parametri

| parametri | Scopo |
|---|---|
| **--mountable** | opzione dice al generatore che si desidera creare un motore "montabile" e isolato nello spazio dei nomi |
| **--pieno** | opzione dice al generatore che vuoi creare un motore, inclusa una struttura a scheletro |

## Osservazioni

I motori sono ottime opzioni per la creazione di plugin riutilizzabili per l'applicazione rails

## Examples

### Esempi famosi sono

Generazione di un motore di blog semplice

```
rails plugin new [engine name] --mountable
```

Esempi di motori famosi sono

Dispositivo (gemma di autenticazione per binari)

Spree (Ecommerce)

Leggi Rails -Engines online: https://riptutorial.com/it/ruby-on-rails/topic/10881/rails--engines

# Capitolo 58: Rails genera comandi

## introduzione

Utilizzo: le `rails generate GENERATOR_NAME [args] [options]` .

Utilizzare le `rails generate` per elencare i generatori disponibili. Alias: `rails g` .

## Parametri

| Parametro | Dettagli |
|---|---|
| `-h` / `--help` | Ottieni aiuto su qualsiasi comando del generatore |
| `-p` / `--pretend` | Modalità di finzione: lancia il generatore ma non crea o modifica alcun file |
| `field:type` | 'nome-campo' è il nome della colonna da creare e 'tipo' è il tipo di dati della colonna. I possibili valori per "tipo" nel `field:type` sono indicati nella sezione Commenti. |

## Osservazioni

I possibili valori per "tipo" nel `field:type` sono:

| Tipo di dati | Descrizione |
|---|---|
| `:string` | Per pezzi di testo più piccoli (di solito ha un limite di caratteri di 255) |
| `:text` | Per pezzi di testo più lunghi, come un paragrafo |
| `:binary` | Memorizzazione di dati tra cui immagini, audio e video |
| `:boolean` | Memorizzazione di valori veri o falsi |
| `:date` | Solo la data |
| `:time` | Solo il tempo |
| `:datetime` | Data e ora |
| `:float` | Conservare i galleggianti senza precisione |
| `:decimal` | Conservare i galleggianti con precisione |
| `:integer` | Memorizzazione di numeri interi |

# Examples

## Rotaie genera modello

Per generare un modello `ActiveRecord` che crea automaticamente le migrazioni di db corrette e i file di test di boilerplate per il modello, immettere questo comando

```
rails generate model NAME column_name:column_type
```

'NOME' è il nome del modello. 'campo' è il nome della colonna nella tabella DB e 'tipo' è il tipo di colonna (es. `name:string` o `body:text` ). Controlla la sezione Commenti per un elenco dei tipi di colonna supportati.

Per configurare chiavi esterne, aggiungere `belongs_to:model_name` .

Quindi, supponiamo che tu voglia configurare un modello `User` che abbia un `username` , un `username` `email` e appartenga a una `School` , dovrai digitare quanto segue

```
rails generate model User username:string email:string school:belongs_to
```

`rails g` è una scorciatoia per `rails generate` . Questo produrrebbe lo stesso risultato

```
rails g model User username:string email:string school:belongs_to
```

## Rails genera migrazione

È possibile generare un file di migrazione delle guide dal terminale utilizzando il seguente comando:

```
rails generate migration NAME [field[:type][:index] field[:type][:index]] [options]
```

Per un elenco di tutte le opzioni supportate dal comando, è possibile eseguire il comando senza argomenti, in quanto i `rails generate migration` .

Ad esempio, se desideri aggiungere i campi `first_name` e `last_name` alla tabella `users` , puoi eseguire:

```
rails generate migration AddNamesToUsers last_name:string first_name:string
```

Rails creerà il seguente file di migrazione:

```
class AddNamesToUsers < ActiveRecord::Migration[5.0]
  def change
    add_column :users, :last_name, :string
    add_column :users, :first_name, :string
  end
end
```

Ora, applica le migrazioni in sospeso al database eseguendo quanto segue nel terminale:

5.0

```
rake db:migrate
```

5.0

```
rails db:migrate
```

**Nota:** per una digitazione ancora inferiore, è possibile sostituire `generate` con `g` .

## Rails Generate Scaffold

***NOTA BENE*** : Scaffolding non è raccomandato a meno che non sia per applicazioni / test CRUD molto convenzionali. Questo può generare un sacco di file (viste / modelli / controller) che non sono necessari nella tua applicazione web causando così mal di testa (cattivo :().

Per generare uno scaffold completamente funzionante per un nuovo oggetto, inclusi modello, controllore, viste, risorse e test, utilizzare il comando `rails g scaffold` .

```
$ rails g scaffold Widget name:string price:decimal
    invoke  active_record
    create    db/migrate/20160722171221_create_widgets.rb
    create    app/models/widget.rb
    invoke    test_unit
    create      test/models/widget_test.rb
    create      test/fixtures/widgets.yml
    invoke  resource_route
     route    resources :widgets
    invoke  scaffold_controller
    create    app/controllers/widgets_controller.rb
    invoke    erb
    create      app/views/widgets
    create      app/views/widgets/index.html.erb
    create      app/views/widgets/edit.html.erb
    create      app/views/widgets/show.html.erb
    create      app/views/widgets/new.html.erb
    create      app/views/widgets/_form.html.erb
    invoke    test_unit
    create      test/controllers/widgets_controller_test.rb
    invoke    helper
    create      app/helpers/widgets_helper.rb
    invoke    jbuilder
    create      app/views/widgets/index.json.jbuilder
    create      app/views/widgets/show.json.jbuilder
    invoke  assets
    invoke    javascript
    create      app/assets/javascripts/widgets.js
    invoke    scss
    create      app/assets/stylesheets/widgets.scss
```

Quindi puoi eseguire `rake db:migrate` per configurare la tabella del database.

Quindi puoi visitare http: // localhost: 3000 / widgets e vedrai uno scaffold CRUD perfettamente

funzionante.

## Rails Genera controller

possiamo creare un nuovo controller con il comando `rails g controller` .

```
$ bin/rails generate controller controller_name
```

Il generatore di controller prevede parametri sotto forma di `generate controller ControllerName action1 action2` .

Di seguito viene creato un controller Greetings con un'azione di Ciao.

```
$ bin/rails generate controller Greetings hello
```

Vedrai il seguente output

```
     create  app/controllers/greetings_controller.rb
      route  get "greetings/hello"
     invoke  erb
     create    app/views/greetings
     create    app/views/greetings/hello.html.erb
     invoke  test_unit
     create    test/controllers/greetings_controller_test.rb
     invoke  helper
     create    app/helpers/greetings_helper.rb
     invoke  assets
     invoke    coffee
     create      app/assets/javascripts/greetings.coffee
     invoke    scss
     create      app/assets/stylesheets/greetings.scss
```

Questo genera il seguente

| File | Esempio |
|---|---|
| File di controllo | `greetings_controller.rb` |
| Vedi il file | `hello.html.erb` |
| File di test funzionale | `greetings_controller_test.rb` |
| Visualizza Helper | `greetings_helper.rb` |
| File JavaScript | `greetings.coffee` |

Aggiungerà anche percorsi per ogni azione in `routes.rb`

Leggi Rails genera comandi online: https://riptutorial.com/it/ruby-on-rails/topic/2540/rails-genera-comandi

# Capitolo 59: Reagire con Rails usando gemme react-rails

## Examples

### Reagire sull'installazione di Rails usando la gemma rails_react

Aggiungi reatt-rails al tuo Gemfile:

```
gem 'react-rails'
```

E installare:

```
bundle install
```

Quindi, esegui lo script di installazione:

```
rails g react:install
```

Questo sarà:

crea un file manifest di components.js e una directory app / assets / javascripts / components /, dove inserirai i tuoi componenti nel tuo application.js:

```
//= require react
//= require react_ujs
//= require components
```

### Utilizzo di react_rails all'interno dell'applicazione

**Build React.js**

Puoi scegliere quale build React.js (sviluppo, produzione, con o senza componenti aggiuntivi) da servire in ogni ambiente aggiungendo una configurazione. Ecco i valori predefiniti:

```
# config/environments/development.rb
MyApp::Application.configure do
  config.react.variant = :development
end


# config/environments/production.rb
MyApp::Application.configure do
  config.react.variant = :production
end
```

Per includere componenti aggiuntivi, usa questa configurazione:

```
MyApp::Application.configure do
  config.react.addons = true # defaults to false
end
```

Dopo aver riavviato il server Rails, // = require react fornirà la build di React.js specificata dalle configurazioni.

react-rails offre alcune altre opzioni per versioni e build di React.js. Vedi VERSIONS.md per maggiori informazioni sull'utilizzo della gemma react-source o sulla tua copia di React.js.

JSX

Dopo aver installato react-rails, riavvia il tuo server. Ora, i file .js.jsx verranno trasformati nella pipeline degli asset.

BabelTransformer opzioni

Puoi usare i trasformatori Babel e plugin personalizzati e passare le opzioni al transpiler Babel aggiungendo le seguenti configurazioni:

```
config.react.jsx_transform_options = {
  blacklist: ['spec.functionName', 'validation.react', 'strict'], # default options
  optional: ["transformerName"],  # pass extra babel options
  whitelist: ["useStrict"] # even more options[enter link description here][1]
}
```

Sotto il cofano, le rotaie reagenti utilizzano il transpiler rubino-babele , per la trasformazione.

## Rendering e montaggio

`react-rails` include un helper di visualizzazione `(react_component)` e un discreto driver JavaScript (react_ujs) che lavorano insieme per mettere i componenti React sulla pagina. Dovresti richiedere il driver UJS nel tuo manifest dopo aver reagito (e dopo i turbolinks se usi Turbolinks).

L'helper di vista inserisce un div nella pagina con la classe di componente richiesta e gli oggetti di scena. Per esempio:

```
<%= react_component('HelloMessage', name: 'John') %>
<!-- becomes: -->
<div data-react-class="HelloMessage" data-react-
props="{&amp;quot;name&amp;quot;:&amp;quot;John&amp;quot;}"></div>
```

Durante il caricamento della pagina, il driver react_ujs eseguirà la scansione della pagina e monterà i componenti utilizzando la classe data-react e gli oggetti reattivi ai dati.

Se Turbolinks è presente, i componenti sono montati sulla pagina: cambia evento e smontato alla pagina: prima di scaricare. Turbolinks> = 2.4.0 è consigliato perché espone eventi migliori.

In caso di chiamate Ajax, il montaggio UJS può essere attivato manualmente chiamando da

javascript:

ReactRailsUJS.mountComponents () La firma dell'helper di visualizzazione è:

```
react_component(component_class_name, props={}, html_options={})
```

`component_class_name` è una stringa che identifica una classe componente accessibile a livello globale. Potrebbe avere punti (es. "MyApp.Header.MenuItem").

```
   `props` is either an object that responds to `#to_json` or an    already-stringified JSON
object (eg, made with Jbuilder, see note    below).
```

`html_options` può includere: `tag:` per utilizzare un elemento diverso da un div per incorporare oggetti di tipo data-react e dati-react-props. `prerender: true` per rendere il componente sul server. `**other` Eventuali altri argomenti (ad es. classe :, id :) vengono passati a content_tag.

Leggi Reagire con Rails usando gemme react-rails online: https://riptutorial.com/it/ruby-on-rails/topic/7032/reagire-con-rails-usando-gemme-react-rails

# Capitolo 60: Rota Best Practice

## Examples

### Non ripeti te stesso (ASCIUTTO)

Per aiutare a mantenere il codice pulito, Rails segue il principio di DRY.

Coinvolge quando è possibile, riutilizzando il maggior numero possibile di codice piuttosto che duplicare codice simile in più punti (ad esempio, utilizzando partial). Questo riduce gli *errori* , mantiene il tuo codice *pulito* e applica il principio della *scrittura del codice una volta* e poi riusandolo. Inoltre, è più facile e più efficiente aggiornare il codice in un'unica posizione piuttosto che aggiornare più parti dello stesso codice. Così rendendo il tuo codice più modulare e robusto.

Anche *Fat Model, Skinny Controller* è DRY, perché scrivi il codice nel tuo modello e nel controller fai solo la chiamata, come:

```
# Post model
scope :unpublished, ->(timestamp = Time.now) { where('published_at IS NULL OR published_at >
?', timestamp) }


# Any controller
def index
    ....
    @unpublished_posts = Post.unpublished
    ....
end

def others
    ...
    @unpublished_posts = Post.unpublished
    ...
end
```

Ciò aiuta anche a condurre una struttura basata su API in cui i metodi interni sono nascosti e le modifiche vengono raggiunte attraverso i parametri di passaggio in modo API.

### Convenzione sulla configurazione

In Rails, ti trovi a guardare *controller, viste e modelli* per il tuo database.

Per ridurre la necessità di una configurazione pesante, Rails implementa le regole per semplificare il lavoro con l'applicazione. Puoi definire le tue regole ma per l'inizio (e per dopo) è una buona idea attenersi alle convenzioni offerte da Rails.

Queste convenzioni accelerano lo sviluppo, mantengono il tuo codice conciso e leggibile e ti consentono una facile navigazione all'interno dell'applicazione.

Le convenzioni abbassano anche le barriere all'ingresso per i principianti. Ci sono così tante

convenzioni in Rails che un principiante non ha nemmeno bisogno di sapere, ma può trarre beneficio dall'ignoranza. È possibile creare grandi applicazioni senza sapere perché tutto sia così com'è.

**Per esempio**

Se si ha una tabella di database chiamata `orders` con l' `id` chiave primaria, il modello corrispondente viene chiamato `order` e il controller che gestisce tutta la logica è denominato `orders_controller` . La vista è divisa in diverse azioni: se il controllore ha un'azione `new` e `edit` , c'è anche una vista `new` e `edit` .

**Per esempio**

Per creare un'app, esegui semplicemente `rails new app_name` . Ciò genererà circa 70 file e cartelle che comprendono l'infrastruttura e le fondamenta della tua app Rails.

Include:

- Cartelle per contenere i modelli (livello database), i controller e le viste
- Cartelle per tenere unit test per la tua applicazione
- Cartelle per conservare le tue risorse web come i file Javascript e CSS
- File predefiniti per risposte 400 HTTP (ad esempio file non trovato)
- Molti altri

## Fat Model, Skinny Controller

"Fat Model, Skinny Controller" si riferisce a come le parti M e C di MVC lavorano idealmente insieme. Vale a dire, qualsiasi logica non correlata alla risposta dovrebbe essere inserita nel modello, idealmente in un buon metodo testabile. Nel frattempo, il controller "magro" è semplicemente una bella interfaccia tra la vista e il modello.

In pratica, questo può richiedere una gamma di diversi tipi di refactoring, ma tutto si riduce a un'idea: spostando qualsiasi logica che non riguarda la risposta al modello (invece del controller), non solo hai promosso il riutilizzo dove possibile ma hai anche reso possibile testare il tuo codice al di fuori del contesto di una richiesta.

Diamo un'occhiata ad un semplice esempio. Di 'che hai un codice come questo:

```
def index
  @published_posts = Post.where('published_at <= ?', Time.now)
  @unpublished_posts = Post.where('published_at IS NULL OR published_at > ?', Time.now)
end
```

Puoi cambiarlo con questo:

```
def index
  @published_posts = Post.published
  @unpublished_posts = Post.unpublished
end
```

Quindi, puoi spostare la logica sul tuo modello di post, dove potrebbe apparire come questa:

```
scope :published, ->(timestamp = Time.now) { where('published_at <= ?', timestamp) }
scope :unpublished, ->(timestamp = Time.now) { where('published_at IS NULL OR published_at >
?', timestamp) }
```

### Attenzione a default_scope

ActiveRecord include `default_scope` , per `default_scope` automaticamente un modello per impostazione predefinita.

```
class Post
  default_scope ->{ where(published: true).order(created_at: :desc) }
end
```

Il codice precedente servirà post che sono già pubblicati quando si esegue una query sul modello.

```
Post.all # will only list published posts
```

Quel mirino, sebbene dall'aspetto innocuo, ha molteplici effetti collaterali nascosti che potresti non volere.

#### `default_scope` e `order`

Dato che hai dichiarato un `order` in `default_scope` , l' `order` chiamata su `Post` verrà aggiunto come ordine aggiuntivo anziché sostituire il valore predefinito.

```
Post.order(updated_at: :desc)
```

```
SELECT "posts".* FROM "posts" WHERE "posts"."published" = 't' ORDER BY "posts"."created_at"
DESC, "posts"."updated_at" DESC
```

Questo probabilmente non è il comportamento che volevi; puoi sovrascriverlo escludendo prima l' `order` dall'ambito

```
Post.except(:order).order(updated_at: :desc)
```

```
SELECT "posts".* FROM "posts" WHERE "posts"."published" = 't' ORDER BY "posts"."updated_at"
DESC
```

## `default_scope` e inizializzazione del modello

Come con qualsiasi altro `ActiveRecord::Relation` , `default_scope` modificherà lo stato predefinito dei modelli inizializzati da esso.

Nell'esempio precedente, `Post` ha `where(published: true)` impostato per impostazione predefinita,

quindi anche i nuovi modelli di `Post` verranno impostati.

```
Post.new # => <Post published: true>
```

### unscoped

`default_scope` può nominalmente essere ovviato chiamando `unscoped` prima, ma questo ha anche effetti collaterali. Prendi, per esempio, un modello STI:

```
class Post < Document
  default_scope ->{ where(published: true).order(created_at: :desc) }
end
```

Per impostazione predefinita, le query su `Post` verranno esaminate per `type` colonne contenenti `'Post'` . Ma non `unscoped` cancellerà questo insieme al tuo `default_scope` , quindi se usi `unscoped` devi ricordarti di tener conto anche di questo.

```
Post.unscoped.where(type: 'Post').order(updated_at: :desc)
```

## unscoped modello e modello

Considera una relazione tra `Post` e `User`

```
class Post < ApplicationRecord
  belongs_to :user
  default_scope ->{ where(published: true).order(created_at: :desc) }
end

class User < ApplicationRecord
  has_many :posts
end
```

Ottenendo un singolo `User` , puoi vedere i post relativi ad esso:

```
user = User.find(1)
user.posts
```

```
SELECT "posts".* FROM "posts" WHERE "posts"."published" = 't' AND "posts"."user_id" = ? ORDER
BY "posts"."created_at" DESC [["user_id", 1]]
```

Ma tu vuoi cancellare il `default_scope` dalla relazione dei `posts` , quindi usi `unscoped`

```
user.posts.unscoped
```

```
SELECT "posts".* FROM "posts"
```

Questo cancella la condizione `user_id` e il `default_scope` .

# Un esempio di caso d'uso per `default_scope`

Nonostante tutto, ci sono situazioni in cui l'uso di `default_scope` è giustificabile.

Si consideri un sistema multi-tenant in cui vengono forniti più sottodomini dalla stessa applicazione ma con dati isolati. Un modo per ottenere questo isolamento è tramite `default_scope` . I lati negativi in altri casi diventano dei lati positivi qui.

```
class ApplicationRecord < ActiveRecord::Base
  def self.inherited(subclass)
    super

    return unless subclass.superclass == self
    return unless subclass.column_names.include? 'tenant_id'

    subclass.class_eval do
      default_scope ->{ where(tenant_id: Tenant.current_id) }
    end
  end
end
```

Tutto ciò che devi fare è impostare `Tenant.current_id` su qualcosa nella fase iniziale della richiesta, e qualsiasi tabella che contenga `tenant_id` verrà automaticamente `tenant_id` senza alcun codice aggiuntivo. I record di istanziazione erediteranno automaticamente l'ID tenant in cui sono stati creati.

La cosa importante di questo caso d'uso è che l'ambito viene impostato una volta per richiesta e non cambia. Gli unici casi in cui è necessario `unscoped` qui sono casi speciali come gli operatori in background che vengono eseguiti al di fuori dell'ambito di una richiesta.

### Non ne hai bisogno (YAGNI)

Se riesci a dire "YAGNI" (non ne avrai bisogno) su una funzione, è meglio non implementarla. È possibile risparmiare molto tempo di sviluppo concentrandosi sulla semplicità. L'implementazione di tali funzionalità può comunque causare problemi:

# I problemi

## Overengineering

Se un prodotto è più complicato di quanto deve essere, è sovradimensionato. Di solito queste funzionalità "non ancora utilizzate" non verranno mai utilizzate nel modo previsto in cui sono state scritte e devono essere nuovamente refactate se mai vengono utilizzate. Le ottimizzazioni premature, in particolare le ottimizzazioni delle prestazioni, portano spesso a decisioni di progettazione che si riveleranno errate in futuro.

## Codice Bloat

Code Bloat significa codice complicato non necessario. Ciò può verificarsi ad esempio per astrazione, ridondanza o applicazione non corretta dei modelli di progettazione. Il codice base diventa difficile da capire, confuso e costoso da mantenere.

## Feature Creep

Feature Creep fa riferimento all'aggiunta di nuove funzionalità che vanno oltre la funzionalità principale del prodotto e portano a una complessità inutilmente elevata del prodotto.

## Lungo tempo di sviluppo

Il tempo che potrebbe essere utilizzato per sviluppare le caratteristiche necessarie è speso per sviluppare funzionalità non necessarie. Il prodotto richiede più tempo per la consegna.

# soluzioni

## BACIO - Resta semplice, stupido

Secondo KISS, la maggior parte dei sistemi funziona meglio se sono progettati in modo semplice. La semplicità dovrebbe essere un obiettivo primario di progettazione per ridurre la complessità. Si può ottenere seguendo il "principio della singola responsabilità", ad esempio.

## YAGNI - Non ne avrai bisogno

Meno è meglio. Pensa a ogni funzione, è davvero necessaria? Se riesci a pensare ad un modo in cui è YAGNI, lascia perdere. È meglio svilupparlo quando è necessario.

## Refactoring continuo

Il prodotto viene costantemente migliorato. Con il refactoring, possiamo fare in modo che il prodotto venga eseguito secondo le migliori pratiche e non degeneri in un lavoro di patch.

### Oggetti di dominio (non più modelli di grasso)

"Fat Model, Skinny Controller" è un ottimo primo passo, ma non si adatta bene quando la base di codice inizia a crescere.

Pensiamo alla singola responsabilità dei modelli. Qual è la singola responsabilità dei modelli? È per mantenere la logica di business? È in grado di mantenere una logica non correlata alla risposta?

No. La sua responsabilità è di gestire lo strato di persistenza e la sua astrazione.

La logica aziendale, così come qualsiasi logica non correlata alla risposta e logica non correlata alla persistenza, dovrebbe andare negli oggetti dominio.

Gli oggetti di dominio sono classi progettate per avere una sola responsabilità nel dominio del problema. Lascia che le tue classi " urlino la loro architettura " per i problemi che risolvono.

In pratica, dovresti cercare modelli magri, punti di vista magri e controller magri. L'architettura della tua soluzione non dovrebbe essere influenzata dal framework che stai scegliendo.

**Per esempio**

Supponiamo che tu sia un marketplace che addebita una commissione fissa del 15% ai tuoi clienti tramite Stripe. Se si addebita una commissione fissa del 15%, ciò significa che la commissione cambia in base all'ammontare dell'ordine poiché Stripe addebita il 2,9% + 30 ¢.

L'importo da addebitare come commissione deve essere: `amount*0.15 - (amount*0.029 + 0.30)` .

Non scrivere questa logica nel modello:

```
# app/models/order.rb
class Order < ActiveRecord::Base
  SERVICE_COMMISSION = 0.15
  STRIPE_PERCENTAGE_COMMISSION = 0.029
  STRIPE_FIXED_COMMISSION = 0.30

  ...

  def commission
    amount*SERVICE_COMMISSION - stripe_commission
  end

  private

  def stripe_commission
    amount*STRIPE_PERCENTAGE_COMMISSION + STRIPE_FIXED_COMMISSION
  end
end
```

Non appena ti integrerai con un nuovo metodo di pagamento, non potrai scalare questa funzionalità all'interno di questo modello.

Inoltre, non appena inizi a integrare più logiche di business, il tuo oggetto `Order` inizierà a perdere coesione .

Preferisci oggetti di dominio, con il calcolo della commissione completamente sottratto alla responsabilità degli ordini persistenti:

```
# app/models/order.rb
class Order < ActiveRecord::Base
  ...
  # No reference to commission calculation
end
```

```
# lib/commission.rb
class Commission
  SERVICE_COMMISSION = 0.15

  def self.calculate(payment_method, model)
    model.amount*SERVICE_COMMISSION - payment_commission(payment_method, model)
  end

  private

  def self.payment_commission(payment_method, model)
    # There are better ways to implement a static registry,
    # this is only for illustration purposes.
    Object.const_get("#{payment_method}Commission").calculate(model)
  end
end

# lib/stripe_commission.rb
class StripeCommission
  STRIPE_PERCENTAGE_COMMISSION = 0.029
  STRIPE_FIXED_COMMISSION = 0.30

  def self.calculate(model)
    model.amount*STRIPE_PERCENTAGE_COMMISSION
      + STRIPE_PERCENTAGE_COMMISSION
  end
end

# app/controllers/orders_controller.rb
class OrdersController < ApplicationController
  def create
    @order = Order.new(order_params)
    @order.commission = Commission.calculate("Stripe", @order)
    ...
  end
end
```

L'utilizzo di oggetti di dominio presenta i seguenti vantaggi architettonici:

- è estremamente facile da testare l'unità, in quanto non sono necessarie fixture o fabbriche per istanziare gli oggetti con la logica.
- funziona con tutto ciò che accetta l' `amount` del messaggio.
- mantiene ogni oggetto del dominio piccolo, con responsabilità chiaramente definite e con maggiore coesione.
- facilmente scalabile con nuovi metodi di pagamento per aggiunta, non modifica .
- blocca la tendenza ad avere un oggetto `User` sempre crescente in ogni applicazione Ruby on Rails.

Personalmente mi piace mettere oggetti di dominio in `lib` . In tal caso, ricordati di aggiungerlo a `autoload_paths` :

```
# config/application.rb
config.autoload_paths << Rails.root.join('lib')
```

Potresti anche preferire creare oggetti di dominio più orientati all'azione, seguendo il modello

Command / Query. In tal caso, inserire questi oggetti in `app/commands` potrebbe essere un posto migliore in quanto tutte le sottodirectory `app` vengono automaticamente aggiunte al percorso di caricamento automatico.

Leggi Rota Best Practice online: https://riptutorial.com/it/ruby-on-rails/topic/1207/rota-best-practice

# Capitolo 61: Rotaie sulla finestra mobile

## introduzione

Questo tutorial inizierà con Docker installato e con un'app Rails

## Examples

### Docker e docker-comporre

Prima di tutto, dovremo creare il nostro `Dockerfile` . Un buon esempio può essere trovato su questo blog di Nick Janetakis.

Questo codice contiene lo script che verrà eseguito sulla nostra finestra mobile al momento dell'avvio. Per questo motivo, stiamo installando tutte le librerie richieste e terminiamo con l'avvio di Puma (server di sviluppo RoR)

```
# Use the barebones version of Ruby 2.3.
FROM ruby:2.3.0-slim

# Optionally set a maintainer name to let people know who made this image.
MAINTAINER Nick Janetakis <nick.janetakis@gmail.com>

# Install dependencies:
# - build-essential: To ensure certain gems can be compiled
# - nodejs: Compile assets
# - libpq-dev: Communicate with postgres through the postgres gem
RUN apt-get update && apt-get install -qq -y --no-install-recommends \
      build-essential nodejs libpq-dev git


# Set an environment variable to store where the app is installed to inside
# of the Docker image. The name matches the project name out of convention only.
ENV INSTALL_PATH /mh-backend
RUN mkdir -p $INSTALL_PATH

# This sets the context of where commands will be running in and is documented
# on Docker's website extensively.
WORKDIR $INSTALL_PATH

# We want binstubs to be available so we can directly call sidekiq and
# potentially other binaries as command overrides without depending on
# bundle exec.
COPY Gemfile* $INSTALL_PATH/

ENV BUNDLE_GEMFILE $INSTALL_PATH/Gemfile
ENV BUNDLE_JOBS 2
ENV BUNDLE_PATH /gembox

RUN bundle install

# Copy in the application code from your work station at the current directory
# over to the working directory.
```

```
COPY . .

# Ensure the static assets are exposed to a volume so that nginx can read
# in these values later.
VOLUME ["$INSTALL_PATH/public"]

ENV RAILS_LOG_TO_STDOUT true


# The default command that gets run will be to start the Puma server.
CMD bundle exec puma -C config/puma.rb
```

Inoltre, useremo la docker-compose, per questo creeremo `docker-compose.yml` . La spiegazione di questo file sarà più un tutorial per la composizione di una finestra mobile di un'integrazione con Rails, che non includerò qui.

```
version: '2'

services:
  backend:
    links:
      - #whatever you need to link like db
    build: .
    command: ./scripts/start.sh
    ports:
      - '3000:3000'
    volumes:
      - .:/backend
    volumes_from:
      - gembox
    env_file:
      - .dev-docker.env
    stdin_open: true
    tty: true
```

Solo con questi due file avrai abbastanza per eseguire `docker-compose up` e riattivare la finestra mobile

Leggi Rotaie sulla finestra mobile online: https://riptutorial.com/it/ruby-on-rails/topic/10933/rotaie-sulla-finestra-mobile

# Capitolo 62: Routing

## introduzione

Il router Rails riconosce gli URL e li invia all'azione di un controllore. Può anche generare percorsi e URL, evitando la necessità di hardcode stringhe nelle viste.

## Osservazioni

"Routing" in generale è il modo in cui gli URL sono "gestiti" dalla tua app. In caso di Rails è in genere il controller e l'azione di quel controller gestirà un particolare URL in arrivo. Nelle app Rails, le rotte vengono di solito collocate nel file `config/routes.rb` .

## Examples

### Routing delle risorse (base)

Le rotte sono definite in `config/routes.rb` . Sono spesso definiti come un gruppo di percorsi correlati, utilizzando le `resources` o i metodi di `resource` .

`resources :users` creano i seguenti sette percorsi, tutti `UsersController` alle azioni di `UsersController` :

```
get       '/users',          to: 'users#index'
post      '/users',          to: 'users#create'
get       '/users/new',      to: 'users#new'
get       '/users/:id/edit', to: 'users#edit'
get       '/users/:id',      to: 'users#show'
patch/put '/users/:id',      to: 'users#update'
delete    '/users/:id',      to: 'users#destroy'
```

I nomi delle azioni sono visualizzati dopo il `#` nel parametro `to` sopra. I metodi con gli stessi nomi devono essere definiti in `app/controllers/users_controller.rb` come segue:

```
class UsersController < ApplicationController
  def index
  end

  def create
  end

  # continue with all the other methods…
end
```

È possibile limitare le azioni generate `only` con o `except` :

```
resources :users, only:   [:show]
resources :users, except: [:show, :index]
```

Puoi visualizzare tutti i percorsi della tua applicazione in qualsiasi momento eseguendo:

5.0

```
$ rake routes
```

5.0

```
$ rake routes
# OR
$ rails routes
```

```
users     GET    /users(.:format)          users#index
          POST   /users(.:format)          users#create
new_user  GET    /users/new(.:format)      users#new
edit_user GET    /users/:id/edit(.:format) users#edit
user      GET    /users/:id(.:format)      users#show
          PATCH  /users/:id(.:format)      users#update
          PUT    /users/:id(.:format)      users#update
          DELETE /users/:id(.:format)      users#destroy
```

Per vedere solo le rotte che si associano a un controller particolare:

5.0

```
$ rake routes -c static_pages
static_pages_home    GET    /static_pages/home(.:format)    static_pages#home
static_pages_help    GET    /static_pages/help(.:format)    static_pages#help
```

5.0

```
$ rake routes -c static_pages
static_pages_home    GET    /static_pages/home(.:format)    static_pages#home
static_pages_help    GET    /static_pages/help(.:format)    static_pages#help

# OR

$ rails routes -c static_pages
static_pages_home    GET    /static_pages/home(.:format)    static_pages#home
static_pages_help    GET    /static_pages/help(.:format)    static_pages#help
```

Puoi cercare attraverso i percorsi usando l'opzione `-g` . Mostra tutte le route che corrispondono parzialmente al nome del metodo helper, al percorso URL o al verbo HTTP:

5.0

```
$ rake routes -g new_user     # Matches helper method
$ rake routes -g POST         # Matches HTTP Verb POST
```

5.0

```
$ rake routes -g new_user     # Matches helper method
$ rake routes -g POST         # Matches HTTP Verb POST
# OR
```

```
$ rails routes -g new_user    # Matches helper method
$ rails routes -g POST        # Matches HTTP Verb POST
```

Inoltre, quando si esegue il server di `rails` in modalità di sviluppo, è possibile accedere a una pagina Web che mostra tutti i percorsi con un filtro di ricerca, abbinato in priorità da cima a fondo, in `<hostname>/rails/info/routes` . Sembrerà così:

| aiutante | Verbale HTTP | Sentiero | Controller # Azione |
|---|---|---|---|
| Percorso / Url | | [Path Match] | |
| users_path | OTTENERE | /users(.:format) | Indice utenti # |
| | INVIARE | /users(.:format) | utenti creano # |
| new_user_path | OTTENERE | /users/new(.:format) | utenti # nuova |
| edit_user_path | OTTENERE | /users/:id/edit(.:format) | gli utenti # modifica |
| user_path | OTTENERE | /users/:id(.:format) | gli utenti # show |
| | PATCH | /users/:id(.:format) | gli utenti # aggiornamento |
| | METTERE | /users/:id(.:format) | gli utenti # aggiornamento |
| | ELIMINA | /users/:id(.:format) | utenti # distruggono |

Le route possono essere dichiarate disponibili solo per i membri (non per le raccolte) utilizzando la `resource` metodo anziché le `resources` in `routes.rb` . Con la `resource` , una route `index` non viene creata per impostazione predefinita, ma solo quando ne viene richiesta esplicitamente una come questa:

```
resource :orders, only: [:index, :create, :show]
```

## vincoli

Puoi filtrare quali percorsi sono disponibili usando i vincoli.

Esistono diversi modi per utilizzare i vincoli, tra cui:

- vincoli di segmento ,
- richiesta basata vincoli
- vincoli avanzati

Ad esempio, un vincolo basato su richiesta per consentire solo a un indirizzo IP specifico di accedere a una rotta:

```
constraints(ip: /127\.0\.0\.1$/) do
  get 'route', to: "controller#action"
```

```
end
```

Vedi altri esempi simili ActionDispatch :: Routing :: Mapper :: Scoping .

Se vuoi fare qualcosa di più complesso puoi usare vincoli più avanzati e creare una classe per avvolgere la logica:

```
# lib/api_version_constraint.rb
class ApiVersionConstraint
  def initialize(version:, default:)
    @version = version
    @default = default
  end

  def version_header
    "application/vnd.my-app.v#{@version}"
  end

  def matches?(request)
    @default || request.headers["Accept"].include?(version_header)
  end
end

# config/routes.rb
require "api_version_constraint"

Rails.application.routes.draw do
  namespace :v1, constraints: ApiVersionConstraint.new(version: 1, default: true) do
    resources :users # Will route to app/controllers/v1/users_controller.rb
  end

  namespace :v2, constraints: ApiVersionConstraint.new(version: 2) do
    resources :users # Will route to app/controllers/v2/users_controller.rb
  end
end
```

### Un modulo, diversi pulsanti di invio

È inoltre possibile utilizzare il valore dei tag di invio di un modulo come vincolo per eseguire il routing a un'azione diversa. Se hai un modulo con più pulsanti di invio (ad esempio "anteprima" e "invia"), puoi catturare questo vincolo direttamente nel tuo `routes.rb` , invece di scrivere javascript per modificare l'URL di destinazione del modulo. Ad esempio con la gem di commit_param_routing puoi sfruttare i binari `submit_tag`

`submit_tag` parametro `submit_tag` Rails consente di modificare il valore del parametro di commit della `submit_tag`

```
# app/views/orders/mass_order.html.erb
<%= form_for(@orders, url: mass_create_order_path do |f| %>
    <!-- Big form here -->
  <%= submit_tag "Preview" %>
  <%= submit_tag "Submit" %>
  # => <input name="commit" type="submit" value="Preview" />
  # => <input name="commit" type="submit" value="Submit" />
  ...
<% end %>
```

```
# config/routes.rb
resources :orders do
  # Both routes below describe the same POST URL, but route to different actions
  post 'mass_order', on: :collection, as: 'mass_order',
    constraints: CommitParamRouting.new('Submit'), action: 'mass_create' # when the user
presses "submit"
  post 'mass_order', on: :collection,
    constraints: CommitParamRouting.new('Preview'), action: 'mass_create_preview' # when the
user presses "preview"
  # Note the `as:` is defined only once, since the path helper is mass_create_order_path for
the form url
  # CommitParamRouting is just a class like ApiVersionContraint
end
```

## Percorsi di ricerca

Rails offre diversi modi per organizzare i tuoi percorsi.

**Ambito per URL** :

```
scope 'admin' do
  get 'dashboard', to: 'administration#dashboard'
  resources 'employees'
end
```

Questo genera i seguenti percorsi

```
get       '/admin/dashboard',          to: 'administration#dashboard'
post      '/admin/employees',          to: 'employees#create'
get       '/admin/employees/new',      to: 'employees#new'
get       '/admin/employees/:id/edit', to: 'employees#edit'
get       '/admin/employees/:id',      to: 'employees#show'
patch/put '/admin/employees/:id',      to: 'employees#update'
delete    '/admin/employees/:id',      to: 'employees#destroy'
```

Potrebbe avere più senso, sul lato server, mantenere alcune viste in una sottocartella diversa, per separare le viste di amministrazione dalle viste degli utenti.

**Ambito per modulo**

```
scope module: :admin do
  get 'dashboard', to: 'administration#dashboard'
end
```

`module` cerca i file del controller sotto la sottocartella del nome specificato

```
get       '/dashboard',          to: 'admin/administration#dashboard'
```

È possibile rinominare il prefisso degli helper del percorso aggiungendo un parametro `as`

```
scope 'admin', as: :administration do
  get 'dashboard'
```

```
end

# => administration_dashboard_path
```

Rails fornisce un modo conveniente per fare tutto quanto sopra, usando il metodo `namespace` . Le seguenti dichiarazioni sono equivalenti

```
namespace :admin do
end

scope 'admin', module: :admin, as: :admin
```

### Ambito dal controller

```
scope controller: :management do
  get 'dashboard'
  get 'performance'
end
```

Questo genera queste rotte

```
get       '/dashboard',          to: 'management#dashboard'
get       '/performance',        to: 'management#performance'
```

### Nidificazione superficiale

Le rotte delle risorse accettano una `:shallow` opzione `:shallow` che aiuta ad accorciare gli URL laddove possibile. Le risorse non devono essere nidificate a più di un livello. Un modo per evitare questo è creare percorsi poco profondi. L'obiettivo è quello di lasciare i segmenti URL della raccolta principale dove non sono necessari. Il risultato finale è che le sole rotte nidificate generate sono per `:index` `:create` e `:new` azioni. Il resto è tenuto nel proprio contesto URL poco profondo. Ci sono due opzioni per l'ambito di percorsi superficiali personalizzati:

- **: shallow_path** : prefissa i percorsi dei membri con un parametro specificato

  ```
  scope shallow_path: "sekret" do
    resources :articles do
      resources :comments, shallow: true
    end
  end
  ```

- **: shallow_prefix** : aggiunge i parametri specificati agli helper specificati

  ```
  scope shallow_prefix: "sekret" do
    resources :articles do
      resources :comments, shallow: true
    end
  end
  ```

Possiamo anche illustrare i percorsi `shallow` più:

```
resources :auctions, shallow: true do
  resources :bids do
   resources :comments
  end
end
```

in alternativa codificato come segue (se sei felice del blocco):

```
resources :auctions do
 shallow do
   resources :bids do
     resources :comments
   end
 end
end
```

I percorsi risultanti sono:

| Prefisso | Verbo | Pattern URI |
|---|---|---|
| bid_comments | OTTENERE | /bids/:bid_id/comments(.:format) |
| | INVIARE | /bids/:bid_id/comments(.:format) |
| new_bid_comment | OTTENERE | /bids/:bid_id/comments/new(.:format) |
| edit_comment | OTTENERE | /comments/:id/edit(.:format) |
| commento | OTTENERE | /comments/:id(.:format) |
| | PATCH | /comments/:id(.:format) |
| | METTERE | /comments/:id(.:format) |
| | ELIMINA | /comments/:id(.:format) |
| auction_bids | OTTENERE | /auctions/:auction_id/bids(.:format) |
| | INVIARE | /auctions/:auction_id/bids(.:format) |
| new_auction_bid | OTTENERE | /auctions/:auction_id/bids/new(.:format) |
| edit_bid | OTTENERE | /bids/:id/edit(.:format) |
| offerta | OTTENERE | /bids/:id(.:format) |
| | PATCH | /bids/:id(.:format) |
| | METTERE | /bids/:id(.:format) |
| | ELIMINA | /bids/:id(.:format) |

| Prefisso | Verbo | Pattern URI |
|---|---|---|
| aste | OTTENERE | /auctions(.:format) |
| | INVIARE | /auctions(.:format) |
| new_auction | OTTENERE | /auctions/new(.:format) |
| edit_auction | OTTENERE | /auctions/:id/edit(.:format) |
| vendita all'asta | OTTENERE | /auctions/:id(.:format) |
| | PATCH | /auctions/:id(.:format) |
| | METTERE | /auctions/:id(.:format) |
| | ELIMINA | /auctions/:id(.:format) |

Se analizzi attentamente le rotte generate, noterai che le parti annidate dell'URL sono incluse solo quando sono necessarie per determinare quali dati visualizzare.

### preoccupazioni

Per evitare la ripetizione in percorsi nidificati, le preoccupazioni forniscono un ottimo modo per condividere risorse comuni che sono riutilizzabili. Per creare un problema, utilizzare la `concern` del metodo all'interno del file `routes.rb` . Il metodo si aspetta un simbolo e un blocco:

```
concern :commentable do
  resources :comments
end
```

Pur non creando alcuna route, questo codice consente di utilizzare l'attributo `:concerns` su una risorsa. L'esempio più semplice sarebbe:

```
resource :page, concerns: :commentable
```

La risorsa nidificata equivalente sarebbe simile a questa:

```
resource :page do
  resource :comments
end
```

Ciò creerebbe, ad esempio, i seguenti percorsi:

```
/pages/#{page_id}/comments
/pages/#{page_id}/comments/#{comment_id}
```

Affinché le preoccupazioni siano significative, devono esserci più risorse che utilizzano la preoccupazione. Risorse aggiuntive potrebbero utilizzare una qualsiasi delle seguenti sintassi per

chiamare la preoccupazione:

```
resource :post, concerns: %i(commentable)
resource :blog do
  concerns :commentable
end
```

## reindirizzamento

È possibile eseguire il reindirizzamento nei percorsi di Rails come segue:

4.0

```
get '/stories', to: redirect('/posts')
```

4.0

```
match "/abc" => redirect("http://example.com/abc")
```

Puoi anche reindirizzare tutte le rotte sconosciute ad un determinato percorso:

4.0

```
match '*path' => redirect('/'), via: :get
# or
get '*path' => redirect('/')
```

4.0

```
match '*path' => redirect('/')
```

## Percorsi membro e raccolta

La definizione di un blocco membro all'interno di una risorsa crea una route che può agire su un singolo membro di quella rotta basata su risorse:

```
resources :posts do
  member do
    get 'preview'
  end
end
```

Questo genera il seguente percorso membro:

```
get '/posts/:id/preview', to: 'posts#preview'
# preview_post_path
```

I percorsi di raccolta consentono di creare percorsi che possono agire su una raccolta di oggetti risorsa:

```
resources :posts do
  collection do
    get 'search'
  end
end
```

Questo genera il seguente percorso di raccolta:

```
get '/posts/search', to: 'posts#search'
# search_posts_path
```

Una sintassi alternativa:

```
resources :posts do
  get 'preview', on: :member
  get 'search',  on: :collection
end
```

## Param di URL con un punto

Se si desidera supportare un parametro url più complesso di un numero id, è possibile che si verifichi un problema con il parser se il valore contiene un punto. Si presuppone che qualsiasi cosa che segue un periodo sia un formato (es. Json, xml).

È possibile aggirare questa limitazione utilizzando un vincolo per *ampliare* l'input accettato.

Ad esempio, se desideri fare riferimento a un record utente tramite indirizzo email nell'URL:

```
resources :users, constraints: { id: /.*/ }
```

## Root route

Puoi aggiungere un percorso della home page alla tua app con il metodo `root` .

```
# config/routes.rb
Rails.application.routes.draw do
  root "application#index"
  # equivalent to:
  # get "/", "application#index"
end

# app/controllers/application_controller.rb
class ApplicationController < ActionController::Base
  def index
    render "homepage"
  end
end
```

E in terminal, i `rake routes` ( `rails routes` di rails in Rails 5) produrranno:

```
root     GET    /         application#index
```

Poiché la home page è in genere il percorso più importante e le route sono prioritarie nell'ordine in cui vengono visualizzate, la route `root` dovrebbe essere in genere la prima nel file dei percorsi.

## Ulteriori azioni RESTful

```
resources :photos do
  member do
    get 'preview'
  end
  collection do
    get 'dashboard'
  end
end
```

Questo crea i seguenti percorsi **oltre alle 7 rotte RESTful predefinite** :

```
get       '/photos/:id/preview',          to: 'photos#preview'
get       '/photos/dashboards',           to: 'photos#dashboard'
```

Se vuoi farlo per linee singole, puoi usare:

```
resources :photos do
  get 'preview',   on: :member
  get 'dashboard', on: :collection
end
```

Puoi anche aggiungere un'azione al `/new` percorso:

```
resources :photos do
  get 'preview', on: :new
end
```

Che creerà:

```
get       '/photos/new/preview',          to: 'photos#preview'
```

Sii consapevole quando aggiungi azioni ai tuoi percorsi RESTful, probabilmente ti manca un'altra risorsa!

## Scope disponibili locales

Se la tua applicazione è disponibile in diverse lingue, di solito mostri la locale corrente nell'URL.

```
scope '/(:locale)', locale: /#{I18n.available_locales.join('|')}/ do
    root 'example#root'
    # other routes
end
```

La tua radice sarà accessibile tramite le impostazioni locali definite in `I18n.available_locales` .

## Montare un'altra applicazione

mount viene utilizzato per montare un'altra applicazione (fondamentalmente un'applicazione rack) o motori di rotaia da utilizzare all'interno dell'applicazione corrente

**sintassi:**

```
mount SomeRackApp, at: "some_route"
```

Ora puoi accedere all'applicazione montata sopra utilizzando helper `some_rack_app_path` o `some_rack_app_url` .

Ma se vuoi rinominare questo nome helper puoi farlo come:

```
mount SomeRackApp, at: "some_route", as: :myapp
```

Questo genererà gli `myapp_path` e `myapp_url` che possono essere utilizzati per navigare in questa app montata.

## Reindirizzamenti e percorsi Wildcard

Se desideri fornire un URL per comodità all'utente, ma mappalo direttamente a un altro che stai già utilizzando. Utilizza un reindirizzamento:

```
# config/routes.rb
TestApp::Application.routes.draw do
  get 'courses/:course_name' => redirect('/courses/%{course_name}/lessons'), :as => "course"
end
```

Bene, è diventato interessante velocemente. Il principio di base qui è utilizzare semplicemente il metodo `#redirect` per inviare una rotta a un'altra rotta. Se il tuo percorso è abbastanza semplice, è un metodo davvero semplice. Ma se vuoi anche inviare i parametri originali, devi fare un po 'di ginnastica catturando il parametro all'interno di `%{here}` . Nota le singole virgolette attorno a tutto.

Nell'esempio sopra, abbiamo anche rinominato la rotta per comodità usando un alias con: come parametro. Questo ci permette di usare quel nome in metodi come gli helper #_path. Di nuovo, prova le tue `$ rake routes` con domande.

## Suddividi i percorsi in più file

Se il tuo file di rotte è enormemente grande, puoi inserire i tuoi percorsi in più file e includere ognuno dei file con il metodo `require_relative` di Ruby:

`config/routes.rb` :

```
YourAppName::Application.routes.draw do
  require_relative 'routes/admin_routes'
  require_relative 'routes/sidekiq_routes'
  require_relative 'routes/api_routes'
```

```
  require_relative 'routes/your_app_routes'
end
```

`config/routes/api_routes.rb` :

```
YourAppName::Application.routes.draw do
  namespace :api do
    # ...
  end
end
```

## Percorsi nidificati

Se si desidera aggiungere rotte nidificate, è possibile scrivere il seguente codice nel file `routes.rb` .

```
resources :admins do
  resources :employees
end
```

Questo genererà i seguenti percorsi:

```
     admin_employees GET      /admins/:admin_id/employees(.:format)            employees#index
                     POST     /admins/:admin_id/employees(.:format)
employees#create
  new_admin_employee GET      /admins/:admin_id/employees/new(.:format)        employees#new
 edit_admin_employee GET      /admins/:admin_id/employees/:id/edit(.:format)   employees#edit
      admin_employee GET      /admins/:admin_id/employees/:id(.:format)        employees#show
                     PATCH    /admins/:admin_id/employees/:id(.:format)
employees#update
                     PUT      /admins/:admin_id/employees/:id(.:format)
employees#update
                     DELETE   /admins/:admin_id/employees/:id(.:format)
employees#destroy
              admins GET      /admins(.:format)                                admins#index
                     POST     /admins(.:format)                                admins#create
           new_admin GET      /admins/new(.:format)                            admins#new
          edit_admin GET      /admins/:id/edit(.:format)                       admins#edit
               admin GET      /admins/:id(.:format)                            admins#show
                     PATCH    /admins/:id(.:format)                            admins#update
                     PUT      /admins/:id(.:format)                            admins#update
                     DELETE   /admins/:id(.:format)                            admins#destroy
```

Leggi Routing online: https://riptutorial.com/it/ruby-on-rails/topic/307/routing

# Capitolo 63: RSpec e Ruby on Rails

## Osservazioni

RSpec è un framework di test per Ruby o, come definito dalla documentazione ufficiale, *RSpec è uno strumento di sviluppo basato sul comportamento per i programmatori Ruby* .

Questo argomento copre le basi dell'utilizzo di RSpec con Ruby on Rails. Per informazioni specifiche su RSpec, visitare l' argomento RSpec .

## Examples

### Installare RSpec

Se si desidera utilizzare il progetto RSpec per un progetto Rails, è necessario utilizzare la gemma `rspec-rails` , che può generare automaticamente helper e file spec (ad esempio, quando si creano modelli, risorse o scaffold utilizzando i `rails generate` ).

Aggiungi `rspec-rails` a entrambi `:development` e `:test` groups nel `Gemfile` :

```
group :development, :test do
  gem 'rspec-rails', '~> 3.5'
end
```

Esegui `bundle` per installare le dipendenze.

Inizializzalo con:

```
rails generate rspec:install
```

Questo creerà una `spec/` cartella per i test, insieme ai seguenti file di configurazione:

- `.rspec` contiene opzioni predefinite per lo strumento `rspec` riga di comando
- `spec/spec_helper.rb` include le opzioni di configurazione di base di RSpec
- `spec/rails_helper.rb` aggiunge ulteriori opzioni di configurazione che sono più specifiche per usare RSpec e Rails insieme.

Tutti questi file sono scritti con impostazioni predefinite per farti iniziare, ma puoi aggiungere funzionalità e modificare le configurazioni in base alle tue esigenze man mano che la tua suite di test cresce.

Leggi RSpec e Ruby on Rails online: https://riptutorial.com/it/ruby-on-rails/topic/5335/rspec-e-ruby-on-rails

# Capitolo 64: Serializzatori di modelli attivi

## introduzione

ActiveModelSerializers, o AMS in breve, porta 'convention over configuration' alla tua generazione JSON. ActiveModelSerializers funziona attraverso due componenti: serializzatori e adattatori. I serializzatori descrivono quali attributi e relazioni devono essere serializzati. Gli adattatori descrivono come gli attributi e le relazioni devono essere serializzati.

## Examples

### Utilizzando un serializzatore

```
class SomeSerializer < ActiveModel::Serializer
  attribute :title, key: :name
  attributes :body
end
```

Leggi Serializzatori di modelli attivi online: https://riptutorial.com/it/ruby-on-rails/topic/9000/serializzatori-di-modelli-attivi

# Capitolo 65: Stati modello: AASM

## Examples

### Stato di base con AASM

Di solito finirai per creare modelli che conterranno uno stato, e lo stato cambierà durante la vita dell'oggetto.

AASM è una libreria di enabler per macchine a stati finiti che può aiutarti a gestire facilmente il processo di progettazione dei tuoi oggetti.

Avere qualcosa del genere nel tuo modello è piuttosto in linea con l'idea di Fat Model, Skinny Controller , una delle best practice di Rails. Il modello è il solo responsabile della gestione del suo stato, dei suoi cambiamenti e della generazione degli eventi innescati da tali cambiamenti.

Per installare, in Gemfile

```
gem 'aasm'
```

Considera un'app in cui l'utente cita un prodotto a un prezzo.

```
class Quote

  include AASM

  aasm do
    state :requested, initial: true  # User sees a product and requests a quote
    state :priced                    # Seller sets the price
    state :payed                     # Buyer pays the price
    state :canceled                  # The buyer is not willing to pay the price
    state :completed                 # The product has been delivered.

    event :price do
        transitions from: requested, to: :priced
    end

    event :pay do
        transitions from: :priced, to: :payed, success: :set_payment_date
    end

    event :complete do
        transitions from: :payed, to: :completed, guard: product_delivered?
    end

    event :cancel do
        transitions from: [:requested, :priced], to: :canceled
        transitions from: :payed, to: canceled, success: :reverse_charges
    end


  end
```

```
  private

  def set_payment_date
    update payed_at: Time.zone.now
  end
end
```

Gli stati della classe Quote possono andare comunque è meglio per il tuo processo.

Puoi pensare agli stati come passati, come nell'esempio precedente o come algo in altri tempi, ad esempio: determinazione dei prezzi, pagamento, consegna, ecc. La denominazione degli stati dipende da te. Da un punto di vista personale, gli stati del passato funzionano meglio perché il tuo stato finale sarà sicuramente un'azione passata e si collegherà meglio con i nomi degli eventi, che verranno spiegati in seguito.

**NOTA:** fai attenzione ai nomi che usi, devi preoccuparti di non usare le parole chiave riservate Ruby o Ruby on Rails, come `valid` , `end` , `being` , etc.

Dopo aver definito gli stati e le transizioni ora possiamo accedere ad alcuni metodi creati da AASM.

Per esempio:

```
Quote.priced  # Shows all Quotes with priced events
quote.priced? # Indicates if that specific quote has been priced
quote.price!  # Triggers the event the would transition from requested to priced.
```

Come puoi vedere l'evento ha delle transizioni, queste transizioni determinano il modo in cui lo stato cambierà sulla chiamata dell'evento. Se l'evento non è valido a causa dello stato corrente, verrà generato un errore.

Gli eventi e le transizioni hanno anche altre callback, per esempio

```
guard: product_delivered?
```

`product_delivered?` metodo che restituirà un valore booleano. Se risulta falso, la transizione non verrà applicata e se non sono disponibili altre transizioni, lo stato non cambierà.

```
success: :reverse_charges
```

Se la traduzione avviene `:reverse_charges` , verrà invocato il metodo `:reverse_charges` .

Ci sono molti altri metodi in AASM con più callback nel processo, ma questo ti aiuterà a creare i tuoi primi modelli con stati finiti.

Leggi Stati modello: AASM online: https://riptutorial.com/it/ruby-on-rails/topic/7826/stati-modello--aasm

# Capitolo 66: Strumenti per l'ottimizzazione e la pulizia del codice Ruby on Rails

## introduzione

Mantenere il proprio codice pulito e organizzato mentre si sviluppa una grande applicazione Rails può essere una vera sfida, anche per uno sviluppatore esperto. Fortunatamente, c'è un'intera categoria di gemme che rendono questo lavoro molto più facile.

## Examples

### Se vuoi mantenere il tuo codice gestibile, sicuro e ottimizzato, guarda alcune gemme per l'ottimizzazione e la pulizia del codice:

#### proiettile

Questo mi ha particolarmente colpito. La gemma dei proiettili ti aiuta a uccidere tutte le query N + 1, così come le relazioni caricate inutilmente ansiose. Una volta installato e avviato la visita di varie rotte in fase di sviluppo, verranno visualizzate caselle di avviso con avvisi che indicano le query del database che devono essere ottimizzate. Funziona fin da subito ed è estremamente utile per ottimizzare l'applicazione.

#### Rota Best Practice

Analizzatore di codice statico per la ricerca di odori di codice specifici di Rails. Offre una varietà di suggerimenti; utilizzare l'accesso all'ambito, limitare i percorsi generati automaticamente, aggiungere indici di database, ecc. Tuttavia, contiene molti suggerimenti che ti offriranno una prospettiva migliore su come ridimensionare il codice e imparare alcune best practice.

#### Rubocop

Un analizzatore di codice statico Ruby che puoi utilizzare per verificare se il tuo codice è conforme alle linee guida del codice comunità di Ruby. La gemma riporta violazioni di stile attraverso la riga di comando, con un sacco di utili codici di refactoring come l'assegnazione di una variabile inutile, l'uso ridondante di Object # to_s nell'interpolazione o anche l'argomento del metodo inutilizzato.

Una cosa buona è che è altamente configurabile, dal momento che l'analizzatore può essere piuttosto irritante se non stai seguendo la guida in stile Ruby al 100% (cioè hai un sacco di spazi bianchi finali o raddoppi le stringhe anche quando non interpolate, ecc.) .

È diviso in 4 sub-analizzatori (chiamati poliziotti): Style, Lint, Metrics and Rails.

Leggi Strumenti per l'ottimizzazione e la pulizia del codice Ruby on Rails online: https://riptutorial.com/it/ruby-on-rails/topic/8713/strumenti-per-l-ottimizzazione-e-la-pulizia-del-codice-ruby-on-rails

# Capitolo 67: Test delle applicazioni Rails

## Examples

### Test unitario

I test unitari eseguono test delle parti dell'applicazione separatamente. di solito un'unità sotto test è una classe o un modulo.

```
let(:gift) { create :gift }

describe '#find' do
  subject { described_class.find(user, Time.zone.now.to_date) }
  it { is_expected.to eq gift }
end
```

fonte

Questo tipo di test è il più diretto e specifico possibile.

### Richiesta di prova

I test di richiesta sono test end to end che imitano il comportamento di un utente.

```
it 'allows the user to set their preferences' do
  check 'Ruby'
  click_on 'Save and Continue'
  expect(user.languages).to eq ['Ruby']
end
```

fonte

Questo tipo di test si concentra sui flussi degli utenti e scorre attraverso tutti gli strati del sistema, talvolta anche rendendo javascript.

Leggi Test delle applicazioni Rails online: https://riptutorial.com/it/ruby-on-rails/topic/7853/test-delle-applicazioni-rails

# Capitolo 68: Transazioni ActiveRecord

## Osservazioni

Le transazioni sono blocchi protettivi in cui le istruzioni SQL sono permanenti solo se riescono tutte come una sola azione atomica. L'esempio classico è un trasferimento tra due account in cui è possibile avere un deposito solo se il ritiro è riuscito e viceversa. Le transazioni rafforzano l'integrità del database e proteggono i dati dagli errori del programma o dalle interruzioni del database. Quindi in pratica dovresti usare i blocchi di transazione ogni volta che hai un numero di istruzioni che devono essere eseguite insieme o non del tutto.

## Examples

### Esempio di base

Per esempio:

```
ActiveRecord::Base.transaction do
  david.withdrawal(100)
  mary.deposit(100)
end
```

Questo esempio prenderà solo denaro da David e lo consegnerà a Mary se né il prelievo né il deposito sollevano un'eccezione. Le eccezioni costringono un ROLLBACK che restituisce il database allo stato prima dell'inizio della transazione. Si noti, tuttavia, che gli oggetti non avranno i loro dati di istanza restituiti allo stato pre-transazione.

### Classi ActiveRecord differenti in una singola transazione

Sebbene il metodo della classe di transazione sia chiamato su alcune classi ActiveRecord, gli oggetti all'interno del blocco di transazione non devono necessariamente essere tutte istanze di quella classe. Questo perché le transazioni sono per connessione al database, non per modello.

In questo esempio, un record di saldo viene salvato a livello di transazione anche se la transazione viene richiamata nella classe Account:

```
Account.transaction do
  balance.save!
  account.save!
end
```

Il metodo di transazione è anche disponibile come metodo di istanza del modello. Ad esempio, puoi anche fare questo:

```
balance.transaction do
  balance.save!
```

```
  account.save!
end
```

## Più connessioni al database

Una transazione agisce su una singola connessione al database. Se si dispone di più database specifici della classe, la transazione non proteggerà l'interazione tra di loro. Una soluzione alternativa è iniziare una transazione su ogni classe di cui modifichiamo i modelli:

```
Student.transaction do
  Course.transaction do
    course.enroll(student)
    student.units += course.units
  end
end
```

Questa è una soluzione scadente, ma le transazioni completamente distribuite vanno oltre lo scopo di ActiveRecord.

## salva e distruggi automaticamente in una transazione

Sia #save che #destroy vengono racchiusi in una transazione che garantisce che qualunque cosa tu faccia in convalide o callback avverrà sotto la sua copertina protetta. Pertanto, è possibile utilizzare le convalide per verificare i valori da cui dipende la transazione oppure è possibile generare eccezioni nei callback per il rollback, inclusi `after_*` callbacks.

Di conseguenza, le modifiche al database non vengono visualizzate al di fuori della connessione fino al completamento dell'operazione. Ad esempio, se si tenta di aggiornare l'indice di un motore di ricerca in `after_save` l'indicizzatore non vedrà il record aggiornato. Il callback `after_commit` è l'unico che viene attivato una volta eseguito il commit dell'aggiornamento.

## callback

Esistono due tipi di callback associati alle transazioni di `after_commit` e `after_rollback` : `after_commit` e `after_rollback` .

`after_commit` callback `after_commit` vengono richiamati su ogni record salvato o distrutto all'interno di una transazione immediatamente dopo il commit della transazione. `after_rollback` callback `after_rollback` vengono richiamati su ogni record salvato o `after_rollback` all'interno di una transazione immediatamente dopo il rollback della transazione o del punto di salvataggio.

Questi callback sono utili per l'interazione con altri sistemi poiché è garantito che la richiamata viene eseguita solo quando il database si trova in uno stato permanente. Ad esempio, `after_commit` è un buon punto per mettere un aggancio per svuotare una cache poiché la sua eliminazione da una transazione potrebbe innescare la rigenerazione della cache prima che il database venga aggiornato.

## Rollback di una transazione

`ActiveRecord::Base.transaction` utilizza l'eccezione `ActiveRecord::Rollback` per distinguere un rollback deliberato da altre situazioni eccezionali. Solitamente, sollevando un'eccezione, il metodo `.transaction` ripristina la transazione del database e trasmette l'eccezione. Ma se si solleva un'eccezione `ActiveRecord::Rollback` , la transazione del database verrà sottoposta a rollback, senza passare l'eccezione.

Ad esempio, è possibile farlo nel controller per eseguire il rollback di una transazione:

```
class BooksController < ActionController::Base
  def create
    Book.transaction do
      book = Book.new(params[:book])
      book.save!
      if today_is_friday?
        # The system must fail on Friday so that our support department
        # won't be out of job. We silently rollback this transaction
        # without telling the user.
        raise ActiveRecord::Rollback, "Call tech support!"
      end
    end
    # ActiveRecord::Rollback is the only exception that won't be passed on
    # by ActiveRecord::Base.transaction, so this line will still be reached
    # even on Friday.
    redirect_to root_url
  end
end
```

Leggi Transazioni ActiveRecord online: https://riptutorial.com/it/ruby-on-rails/topic/4688/transazioni-activerecord

# Capitolo 69: Transazioni ActiveRecord

## introduzione

Le transazioni ActiveRecord sono blocchi protettivi in cui la sequenza di query di record attive è permanente solo se tutte possono avere successo come un'unica azione atomica.

## Examples

### Iniziare con le transazioni dei record attivi

Le Transazioni record attive possono essere applicate alle classi del modello e alle istanze del modello, gli oggetti all'interno del blocco della transazione non devono necessariamente essere istanze della stessa classe. Questo perché le transazioni sono per connessione al database, non per modello. Per esempio:

```
User.transaction do
  account.save!
  profile.save!
  print "All saves success, returning 1"
  return 1
end
rescue_from ActiveRecord::RecordInvalid do |exception|
  print "Exception thrown, transaction rolledback"
  render_error "failure", exception.record.errors.full_messages.to_sentence
end
```

L'uso di save with a bang assicura che la transazione venga automaticamente ripristinata quando viene generata l'eccezione e dopo il rollback, il controllo passa al blocco di salvataggio per l'eccezione. **Assicurati di salvare le eccezioni generate dal salvataggio! in Transaction Block.**

Se non vuoi usare save !, puoi aumentare manualmente `raise ActiveRecord::Rollback` quando il salvataggio fallisce. Non è necessario gestire questa eccezione. Quindi eseguirà il rollback della transazione e prenderà il controllo sull'istruzione successiva dopo il blocco della transazione.

```
   User.transaction do
      if account.save && profile.save
        print "All saves success, returning 1"
        return 1
      else
        raise ActiveRecord::Rollback
      end
    end
    print "Transaction Rolled Back"
```

Leggi Transazioni ActiveRecord online: https://riptutorial.com/it/ruby-on-rails/topic/9326/transazioni-activerecord

# Capitolo 70: Turbolinks

## introduzione

Turbolinks è una libreria javascript che rende più veloce la navigazione nella tua applicazione web. Quando segui un link, Turbolinks recupera automaticamente la pagina, esegue lo scambio nel suo <body> e unisce il suo <head>, il tutto senza incorrere nel costo di un caricamento di una pagina intera.

## Osservazioni

Come sviluppatore di rotaie, probabilmente interagirai minimamente con i turbolink durante lo sviluppo. È, tuttavia, una libreria importante con cui familiarizzare perché può essere la causa di alcuni bug difficili da trovare.

# Key takeaway:

- Associa ai `turbolinks:load` evento invece dell'evento `document.ready`
- Utilizzare l'attributo `data-turbolinks-false` per disabilitare la funzionalità di turbolink su base per-link.
- Utilizza l'attributo `data-turbolinks-permanent` per mantenere gli elementi tra i carichi di pagina e per evitare i bug relativi alla cache.

Per un trattamento più approfondito dei turbolinks, visitare il repository github ufficiale .

Il credito per gran parte di questa documentazione va a coloro che hanno redatto la documentazione dei turbolinks sul repository github.

## Examples

### Associazione al concetto di turbolink di un caricamento della pagina

Con i turbolinks, l'approccio tradizionale all'utilizzo:

```
$(document).ready(function() {
  // awesome code
});
```

non funzionerà Durante l'uso di turbolinks, l'evento `$(document).ready()` solo una volta: sul caricamento della pagina iniziale. Da quel momento in poi, ogni volta che un utente fa clic su un link sul tuo sito web, i turbolinks intercetteranno l'evento click link e faranno una richiesta ajax per sostituire il tag <body> e unire i tag <head>. L'intero processo attiva la nozione di "visita" nella terra dei turbolinks. Pertanto, anziché utilizzare la sintassi tradizionale `document.ready()` , dovrai associare l'evento di visita di turbolink in questo modo:

```
// pure js
document.addEventListener("turbolinks:load", function() {
  // awesome code
});

// jQuery
$(document).on('turbolinks:load', function() {
  // your code
});
```

### Disabilita i turbolinks su link specifici

È molto facile disabilitare i turbolinks su collegamenti specifici. Secondo la documentazione ufficiale dei turbolinks :

> I turbolink possono essere disabilitati per ogni collegamento annotando un link o uno dei suoi antenati con data-turbolinks = "false".

## Esempi:

```
// disables turbolinks for this one link
<a href="/" data-turbolinks="false">Disabled</a>

// disables turbolinks for all links nested within the div tag
<div data-turbolinks="false">
  <a href="/">I'm disabled</a>
  <a href="/">I'm also disabled</a>
</div>

// re-enable specific link when ancestor has disabled turbolinks
<div data-turbolinks="false">
  <a href="/">I'm disabled</a>
  <a href="/" data-turbolinks="true">I'm re-enabled</a>
</div>
```

### Comprensione delle visite alle applicazioni

Le visite alle applicazioni vengono avviate facendo clic su un collegamento abilitato per Turbolinks o chiamando a livello di programmazione

```
Turbolinks.visit(location)
```

Per impostazione predefinita, la funzione di visita utilizza l'azione 'avanzata'. Più comprensibilmente, il comportamento predefinito per la funzione di visita è quello di avanzare alla pagina indicata dal parametro "posizione". Ogni volta che una pagina viene visitata, i turbolinks inseriscono una nuova voce nella cronologia del browser utilizzando `history.pushState` . La cronologia è importante perché i turbolinks cercheranno di utilizzare la cronologia per caricare le pagine dalla cache ogni volta che è possibile. Ciò consente un rendering della pagina estremamente veloce per le pagine visitate di frequente.

Tuttavia, se si desidera visitare una posizione senza inserire alcuna cronologia nello stack, è possibile utilizzare l'azione 'replace' nella funzione di visita in questo modo:

```
// using links
<a href="/edit" data-turbolinks-action="replace">Edit</a>

// programatically
Turbolinks.visit("/edit", { action: "replace" })
```

Questo sostituirà la cima dello stack cronologico con la nuova pagina in modo che il numero totale di elementi nello stack rimanga invariato.

Esiste anche un'azione di "ripristino" che aiuta nelle visite di ripristino , le visite che si verificano a seguito dell'utente che fa clic sul pulsante Avanti o sul pulsante Indietro sul proprio browser. Turbolinks gestisce internamente questi tipi di eventi e consiglia agli utenti di non manomettere manualmente il comportamento predefinito.

### Annullare le visite prima che inizino

Turbolinks fornisce un listener di eventi che può essere utilizzato per interrompere le visite. Ascolta i `turbolinks:before-visit` evento `turbolinks:before-visit` per essere avvisato quando una visita sta per iniziare.

Nel gestore eventi, puoi usare:

```
// pure javascript
event.data.url
```

o

```
// jQuery
$event.originalEvent.data.url
```

per recuperare la posizione della visita. La visita può quindi essere cancellata chiamando:

```
event.preventDefault()
```

## NOTA:

Secondo i documenti ufficiali di turbolinks :

> Le visite di restauro non possono essere annullate e non si attivano i turbolinks: prima visita.

### Elementi persistenti tra i carichi di pagina

Si consideri la seguente situazione: Immagina di essere lo sviluppatore di un sito Web di social

media che consente agli utenti di essere amici di altri utenti e che utilizza i turbolinks per velocizzare il caricamento della pagina. In alto a destra di ogni pagina del sito, c'è un numero che indica il numero totale di amici che un utente ha attualmente. Immagina di usare il tuo sito e di avere 3 amici. Ogni volta che viene aggiunto un nuovo amico, è disponibile un javascript che aggiorna il contatore degli amici. Immagina di aver appena aggiunto un nuovo amico e che il tuo javascript sia eseguito correttamente e aggiornato il conteggio degli amici nella parte in alto a destra della pagina per eseguire il rendering 4. Ora, immagina di fare clic sul pulsante Indietro del browser. Quando la pagina viene caricata, noti che il contatore degli amici dice 3 anche se hai quattro amici.

Questo è un problema relativamente comune e uno per il quale i turbolinks hanno fornito una soluzione. Il motivo per cui si verifica il problema è perché i turbolinks caricano automaticamente le pagine dalla cache quando un utente fa clic sul pulsante Indietro. La pagina memorizzata nella cache non verrà sempre aggiornata con il database.

Per risolvere questo problema, immagina di eseguire il rendering del numero di amici all'interno di un tag  con un ID di "conteggio degli amici":

```
<div id="friend-count" data-turbolinks-permanent>3 friends</div>
```

Aggiungendo l'attributo `data-turbolinks-permanent` , stai dicendo ai turbolinks di mantenere determinati elementi attraverso i carichi di pagina. I documenti ufficiali dicono :

> Designare elementi permanenti dando loro un ID HTML e annotandoli con data-turbolinks-permanenti. Prima di ogni rendering, Turbolinks associa tutti gli elementi permanenti con ID e li trasferisce dalla pagina originale alla nuova pagina, preservando i dati e i listener di eventi.

Leggi Turbolinks online: https://riptutorial.com/it/ruby-on-rails/topic/9331/turbolinks

# Capitolo 71: Upload di file

## Examples

### Caricamento di singoli file usando Carrierwave

Iniziare a usare File Uploads in Rails è abbastanza semplice, la prima cosa che devi fare è scegliere il plugin per la gestione dei caricamenti. Le forze più comuni sono **Carrierwave** e **Paperclip** . Entrambi sono simili per funzionalità e ricchi di documentazione su

Diamo un'occhiata all'esempio con l'immagine di caricamento avatar semplice con Carrierwave

Dopo l' `bundle install` Carrierwave, digitare nella console

```
$ rails generate uploader ProfileUploader
```

Questo creerà un file di configurazione situato in */app/uploaders/profile_uploader.rb*

Qui puoi impostare lo spazio di archiviazione (ad es. Locale o cloud), applicare estensioni per manipolazioni di immagini (ad esempio generare thumbs tramite MiniMagick) e impostare l'elenco bianco di estensioni lato server

Successivamente, crea nuova migrazione con string tipe per user_pic e installa uploader per esso nel modello *user.rb.*

```
mount_uploader :user_pic, ProfileUploader
```

Successivamente, visualizza un modulo per caricare l'avatar (potrebbe essere una vista di modifica per l'utente)

```
<% form_for @user, html: { multipart: true } do |f| %>
    <%= f.file_field :user_pic, accept: 'image/png, image/jpg' %>
    <%= f.submit "update profile pic", class: "btn" %>
<% end %>
```

Assicurati di includere {multipart: true} nel modulo d'ordine in grado di elaborare i caricamenti. Accept è un optional per impostare la white-list dell'estensione lato client.

Per visualizzare un avatar, fallo semplicemente

```
<%= image_tag @user.user_pic.url %>
```

### Modello annidato: più caricamenti

Se si desidera creare più caricamenti, la prima cosa che si potrebbe voler fare è creare un nuovo modello e impostare le relazioni

Supponiamo che tu desideri immagini multiple per il modello del prodotto. Crea un nuovo modello e `belongs_to` al modello principale

```
rails g model ProductPhoto

#product.rb
has_many :product_photos, dependent: :destroy
accepts_nested_attributes_for :product_photos

#product_photo.rb
belongs_to :product
mount_uploader :image_url, ProductPhotoUploader # make sure to include uploader (Carrierwave
example)
```

*accept_nested_attributes_for* è necessario, perché ci consente di creare un modulo annidato, in modo che possiamo caricare nuovo file, cambiare il nome del prodotto e impostare il prezzo da un unico modulo

Successivamente, crea un modulo in una vista (modifica / crea)

```
    <%= form_for @product, html: { multipart: true } do |product|%>

        <%= product.text_field :price # just normal type of field %>

        <%= product.fields_for :product_photos do |photo| # nested fields %>
            <%= photo.file_field :image, :multiple => true, name:
"product_photos[image_url][]" %>
        <% end %>
        <%= p.submit "Update", class: "btn" %>
    <% end %>
```

Il controller non è niente di speciale, se non vuoi crearne uno nuovo, creane uno nuovo all'interno del controller del prodotto

```
  # create an action
  def upload_file
    printer = Product.find_by_id(params[:id])
    @product_photo = printer.prodcut_photos.create(photo_params)
  end

  # strong params
  private
    def photo_params
      params.require(:product_photos).permit(:image)
    end
```

Mostra tutte le immagini in una vista

```
    <% @product.product_photos.each do |i| %>
        <%= image_tag i.image.url, class: 'img-rounded' %>
    <% end %>
```

Leggi Upload di file online: https://riptutorial.com/it/ruby-on-rails/topic/2831/upload-di-file

# Capitolo 72: Utilizzo di Google Maps con Rails

## Examples

### Aggiungi il tag javascript di google maps all'intestazione del layout

Per fare in modo che google maps funzioni correttamente con i turbolinks , aggiungi il tag javascript direttamente all'intestazione del layout piuttosto che includerlo in una vista.

```
# app/views/layouts/my_layout.html.haml
!!!
%html{:lang => 'en'}
  %head
    - # ...
    = google_maps_api_script_tag
```

`google_maps_api_script_tag` è meglio definito in un helper.

```
# app/helpers/google_maps_helper.rb
module GoogleMapsHelper
  def google_maps_api_script_tag
    javascript_include_tag google_maps_api_source
  end

  def google_maps_api_source
    "https://maps.googleapis.com/maps/api/js?key=#{google_maps_api_key}"
  end

  def google_maps_api_key
    Rails.application.secrets.google_maps_api_key
  end
end
```

Puoi registrare la tua domanda con google e ottenere la tua chiave API nella console di Google API . Google ha una breve guida su come richiedere una chiave API per le API di javascript di google maps .

La chiave API è memorizzata nel file `secrets.yml` :

```
# config/secrets.yml
development:
  google_maps_api_key: '...'
  # ...
production:
  google_maps_api_key: '...'
  # ...
```

Non dimenticate di aggiungere `config/secrets.yml` al `.gitignore` di file e makre sicuri di non commettere la chiave API al repository.

## Geocodifica il modello

Supponiamo che i tuoi utenti e / o gruppi abbiano profili e che tu voglia visualizzare i campi del profilo dell'indirizzo su una mappa di google.

```
# app/models/profile_fields/address.rb
class ProfileFields::Address < ProfileFields::Base
  # Attributes:
  # label, e.g. "Work address"
  # value, e.g. "Willy-Brandt-Straße 1\n10557 Berlin"
end
```

Un ottimo modo per geocodificare gli indirizzi, ovvero fornire `longitude` e `latitude` è la gemma del geocoder .

Aggiungi geocoder al tuo `Gemfile` ed esegui `bundle` per installarlo.

```
# Gemfile
gem 'geocoder', '~> 1.3'
```

Aggiungi colonne di database per `latitude` e `longitude` per salvare la posizione nel database. Questo è più efficiente dell'interrogazione del servizio di geocoding ogni volta che è necessario il percorso. È più veloce e non stai raggiungendo il limite della query così velocemente.

```
➜ bin/rails generate migration add_latitude_and_longitude_to_profile_fields \
  latitude:float longitude:float
➜ bin/rails db:migrate  # Rails 5, or:
➜ rake db:migrate       # Rails 3, 4
```

Aggiungi il meccanismo di geocoding al tuo modello. In questo esempio, la stringa di indirizzo è memorizzata nell'attributo `value` . Configura la geocodifica da eseguire quando il record è cambiato e solo se è presente un valore:

```
# app/models/profile_fields/address.rb
class ProfileFields::Address < ProfileFields::Base
  geocoded_by :value
  after_validation :geocode, if: ->(address_field){
    address_field.value.present? and address_field.value_changed?
  }

end
```

Per impostazione predefinita, geocoder utilizza google come servizio di ricerca. Ha molte caratteristiche interessanti come i calcoli della distanza o la ricerca di prossimità. Per maggiori informazioni, date un'occhiata al README del geocoder .

## Mostra indirizzi su una mappa di google nella vista profilo

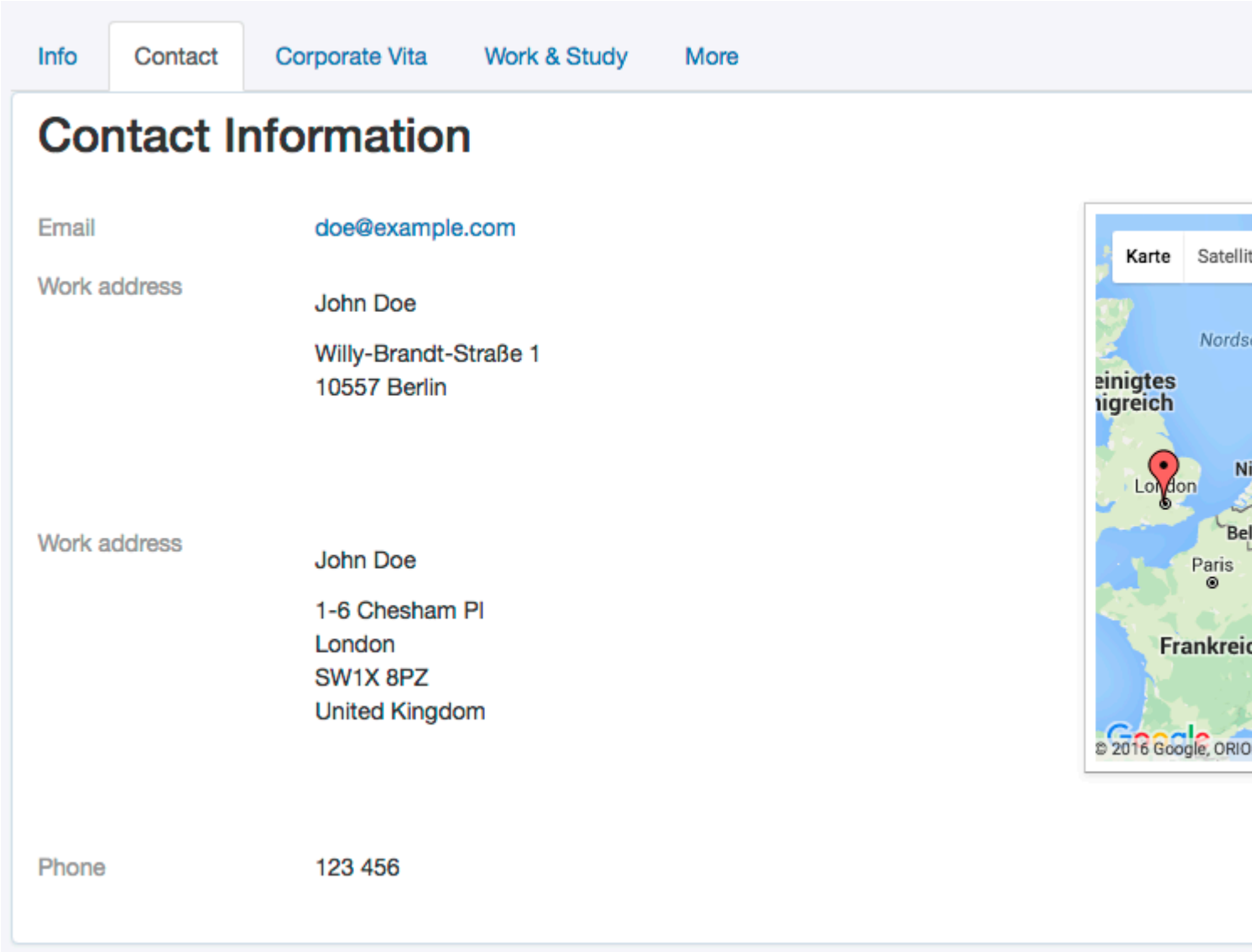
Nella vista profilo, mostra i campi del profilo di un utente o di un gruppo in un elenco, nonché i campi degli indirizzi su una mappa di google.

```
- # app/views/profiles/show.html.haml
%h1 Contact Information
.profile_fields
  = render @profile_fields
.google_map{data: address_fields: @address_fields.to_json }
```

I `@profile_fields` e `@address_fields` appropriati sono impostati nel controller:

```
# app/controllers/profiles_controller.rb
class ProfilesController < ApplicationController
  def show
    # ...
    @profile_fields = @user_or_group.profile_fields
    @address_fields = @profile_fields.where(type: 'ProfileFields::Address')
  end
end
```

Inizializza la mappa, posiziona i marcatori, imposta lo zoom e altre impostazioni della mappa con javascript.




## Imposta i marcatori sulla mappa con javascript

Supponiamo che esista un div `.google_map` , che diventerà la mappa e che abbia i campi degli indirizzi da mostrare come marcatori come attributo dei `data` .

Per esempio:

```
<!-- http://localhost:3000/profiles/123 -->
<div class="google_map" data-address-fields="[
  {label: 'Work address', value: 'Willy-Brandt-Straße 1\n10557 Berlin',
  position: {lng: ..., lat: ...}},
  ...
]"></div>
```

Per utilizzare l'evento `$(document).ready` con i turbolinks senza gestire manualmente gli eventi turbolinks, utilizzare la gem jquery.turbolinks .

Se si desidera eseguire altre operazioni con la mappa, in seguito, ad esempio il filtro o le finestre informative, è conveniente che la mappa sia gestita da una classe di script caffè .

```
# app/assets/javascripts/google_maps.js.coffee
window.App = {} unless App?
class App.GoogleMap
  constructor: (map_div)->
    # TODO: initialize the map
    # TODO: set the markers
```

Quando si utilizzano diversi file di script caffè, che sono assegnati in modo predefinito per nome, è conveniente definire uno spazio dei nomi di `App` globale, condiviso da tutti i file di script caffè.

Quindi, passa attraverso (possibilmente diversi) `.google_map` div e crea un'istanza della classe `App.GoogleMap` per ognuno di essi.

```
# app/assets/javascripts/google_maps.js.coffee
# ...
$(document).ready ->
  App.google_maps = []
  $('.google_map').each ->
    map_div = $(this)
    map = new App.GoogleMap map_div
    App.google_maps.push map
```

## Inizializza la mappa utilizzando una classe di script caffè.

Fornita una classe di script per caffè `App.GoogleMap` , la mappa di google può essere inizializzata in questo modo:

```
# app/assets/javascripts/google_maps.js.coffee
# ...
class App.GoogleMap
  map_div: {}
  map: {}

  constructor: (map_div)->
    @map_div = map_div
```

```
    @init_map()
    @reference_the_map_as_data_attribute

  # To access the GoogleMap object or the map object itself
  # later via the DOM, for example
  #
  #     $('.google_map').data('GoogleMap')
  #
  # store references as data attribute of the map_div.
  #
  reference_the_map_as_data_attribute: ->
    @map_div.data 'GoogleMap', this
    @map_div.data 'map', @map

  init_map: ->
    @map = new google.maps.Map(@dom_element, @map_configuration) if google?

  # `@map_div` is the jquery object. But google maps needs
  # the real DOM element.
  #
  dom_element: ->
    @map_div.get(0)

  map_configuration: -> {
    scrollWheel: true
  }
```

Per ulteriori informazioni sulle possibili opzioni `map_configuration` , dai un'occhiata alla documentazione di MapOptions di google e alla loro guida per aggiungere elementi di controllo .

Per riferimento, la classe `google.maps.Map` è ampiamente documentata qui .

## Inizializza gli indicatori di mappa utilizzando una classe di script caffè

A condizione che una classe di script caffè `App.GoogleMap` e le informazioni sul marker vengano memorizzate nell'attributo `data-address-fields` del div `.google_map` , gli indicatori di mappa possono essere inizializzati sulla mappa in questo modo:

```
# app/assets/javascripts/google_maps.js.coffee
# ...
class App.GoogleMap
  # ...
  markers: []

  constructor: (map_div)->
    # ...
    @init_markers()

  address_fields: ->
    @map_div.data('address-fields')

  init_markers: ->
    self = this  # to reference the instance as `self` when `this` is redefined.
    self.markers = []
    for address_field in self.address_fields()
      marker = new google.maps.Marker {
        map: self.map,
```

```
        position: {
          lng: address_field.longitude,
          lat: address_field.latitude
        },
        # # or, if `position` is defined in `ProfileFields::Address#as_json`:
        # position: address_field.position,
        title: address_field.value
      }
      self.markers.push marker
```

Per ulteriori informazioni sulle opzioni dei marker, dai un'occhiata alla documentazione di MarkerOptions di google e alla loro guida ai marcatori .

### Zoom automatico di una mappa utilizzando una classe di script caffè

Fornita una classe di script per caffè `App.GoogleMap` con la `google.maps.Map` memorizzata come `@map` e i `google.maps.Marker` s memorizzati come `@markers` , la mappa può essere ingrandita automaticamente, ovvero regolata affinché tutti i marcatori siano visibili, come questo : sulla mappa in questo modo:

```
# app/assets/javascripts/google_maps.js.coffee
# ...
class App.GoogleMap
  # ...
  bounds: {}

  constructor: (map_div)->
    # ...
    @auto_zoom()

  auto_zoom: ->
    @init_bounds()
    # TODO: Maybe, adjust the zoom to have a maximum or
    # minimum zoom level, here.

  init_bounds: ->
    @bounds = new google.maps.LatLngBounds()
    for marker in @markers
      @bounds.extend marker.position
    @map.fitBounds @bounds
```

Per saperne di più sui limiti, dai un'occhiata alla documentazione di LatLngBounds di google.

### Esporre le proprietà del modello come json

Per visualizzare i campi del profilo dell'indirizzo come marcatori su una mappa di google, gli oggetti del campo indirizzo devono essere passati come oggetti json a javascript.

# Attributi regolari del database

Quando si chiama `to_json` su un oggetto `ApplicationRecord` , gli attributi del database vengono automaticamente esposti.

Dato un modello `ProfileFields::Address` con gli attributi `label` , `value` , `longitude` e `latitude` , `address_field.as_json` risulta in un `Hash` , ad es. Rappresentazione,

```
address_field.as_json  # =>
  {label: "Work address", value: "Willy-Brandt-Straße 1\n10557 Berlin",
    longitude: ..., latitude: ...}
```

che viene convertito in una stringa json di `to_json` :

```
address_field.to_json  # =>
  "{\"label\":\"Work address\",\"value\":\"Willy-Brandt-Straße 1\\n
    10557 Berlin\",\"longitude\":...,\"latitude\":...}"
```

Ciò è utile perché consente di utilizzare l' `label` e il `value` seguito in javascript, ad esempio per mostrare suggerimenti sugli strumenti per gli indicatori di mappa.

# Altri attributi

Altri attributi virtuali possono essere esposti sovrascrivendo il metodo `as_json` .

Ad esempio, per esporre un attributo `title` , `as_json` nell'hash unito `as_json` :

```
# app/models/profile_fields/address.rb
class ProfileFields::Address < ProfileFields::Base
  # ...

  # For example: "John Doe, Work address"
  def title
    "#{self.parent.name}, #{self.label}"
  end

  def as_json
    super.merge {
      title: self.title
    }
  end
end
```

L'esempio precedente utilizza `super` per chiamare il metodo `as_json` originale, che restituisce l'hash dell'attributo originale dell'oggetto e lo unisce con l'hash della posizione richiesta.

Per capire la differenza tra `as_json` e `to_json` , dai un'occhiata a questo post sul blog di jjulian .

# Posizione

Per rendere i marcatori, l'API di google maps, per impostazione predefinita, richiede un hash di `position` che ha la longitudine e la latitudine memorizzate come `lng` e `lat` rispettivamente.

Questo hash di posizione può essere creato in javascript, in seguito, o qui quando si definisce la

rappresentazione json del campo dell'indirizzo:

Per fornire questa `position` come attributo json del campo dell'indirizzo, basta sovrascrivere il metodo `as_json` sul modello.

```
# app/models/profile_fields/address.rb
class ProfileFields::Address < ProfileFields::Base
  # ...

  def as_json
    super.merge {
      # ...
      position: {
        lng: self.longitude,
        lat: self.latitude
      }
    }
  end
end
```

Leggi Utilizzo di Google Maps con Rails online: https://riptutorial.com/it/ruby-on-rails/topic/2828/utilizzo-di-google-maps-con-rails

# Capitolo 73: Validazioni ActiveRecord

## Examples

### Convalida della numericità di un attributo

Questa convalida limita l'inserimento di soli valori numerici.

```
class Player < ApplicationRecord
  validates :points, numericality: true
  validates :games_played, numericality: { only_integer: true }
end
```

Inoltre `:only_integer` , questo helper accetta anche le seguenti opzioni per aggiungere vincoli a valori accettabili:

- `:greater_than` - Specifica che il valore deve essere maggiore del valore fornito. Il messaggio di errore predefinito per questa opzione è "deve essere maggiore di% {count}".
- `:greater_than_or_equal_to` - Specifica che il valore deve essere maggiore o uguale al valore fornito. Il messaggio di errore predefinito per questa opzione è "deve essere maggiore o uguale a% {count}".
- `:equal_to` - Specifica che il valore deve essere uguale al valore fornito. Il messaggio di errore predefinito per questa opzione è "deve essere uguale a% {count}".
- `:less_than` - Specifica che il valore deve essere inferiore al valore fornito. Il messaggio di errore predefinito per questa opzione è "deve essere inferiore a% {count}".
- `:less_than_or_equal_to` - Specifica che il valore deve essere minore o uguale al valore fornito. Il messaggio di errore predefinito per questa opzione è "deve essere minore o uguale a% {count}".
- `:other_than` - Specifica che il valore deve essere diverso dal valore fornito. Il messaggio di errore predefinito per questa opzione è "deve essere diverso da% {count}".
- `:odd` - Specifica che il valore deve essere un numero dispari se impostato su true. Il messaggio di errore predefinito per questa opzione è "deve essere dispari".
- `:even` - Specifica che il valore deve essere un numero pari se impostato su true. Il messaggio di errore predefinito per questa opzione è "deve essere pari".

  Per impostazione predefinita, la numericità non consente valori nulli. Puoi usare allow_nil: true per permetterlo.

### Convalida l'unicità di un attributo

Questo helper convalida che il valore dell'attributo è univoco proprio prima che l'oggetto venga salvato.

```
class Account < ApplicationRecord
  validates :email, uniqueness: true
end
```

Esiste un'opzione `:scope` che è possibile utilizzare per specificare uno o più attributi utilizzati per limitare il controllo di unicità:

```
class Holiday < ApplicationRecord
  validates :name, uniqueness: { scope: :year,
    message: "should happen once per year" }
end
```

Esiste anche un'opzione `:case_sensitive` che è possibile utilizzare per definire se il vincolo di univocità sarà sensibile al maiuscolo o minuscolo. Questa opzione è impostata su `true` .

```
class Person < ApplicationRecord
  validates :name, uniqueness: { case_sensitive: false }
end
```

## Convalida presenza di un attributo

Questo helper convalida che gli attributi specificati non sono vuoti.

```
class Person < ApplicationRecord
  validates :name, presence: true
end

Person.create(name: "John").valid? # => true
Person.create(name: nil).valid? # => false
```

È possibile utilizzare l'helper `absence` per convalidare che gli attributi specificati sono assenti. Usa il `present?` metodo per verificare valori nulli o vuoti.

```
class Person < ApplicationRecord
  validates :name, :login, :email, absence: true
end
```

**Nota:** se l'attributo è di tipo `boolean` , non è possibile utilizzare la normale convalida di presenza (l'attributo non sarebbe valido per un valore `false` ). È possibile ottenere questo risultato utilizzando una convalida di inclusione:

```
validates :attribute, inclusion: [true, false]
```

## Saltare le convalide

Usa i seguenti metodi se vuoi saltare le convalide. Questi metodi salveranno l'oggetto nel database anche se non è valido.

- decrementare!
- decrement_counter
- incremento!
- increment_counter
- alternare!

- toccare
- aggiorna tutto
- update_attribute
- update_column
- update_columns
- update_counters

È inoltre possibile saltare la convalida durante il salvataggio passando `validate` come argomento per `save`

```
User.save(validate: false)
```

## Convalida della lunghezza di un attributo

```
class Person < ApplicationRecord
  validates :name, length: { minimum: 2 }
  validates :bio, length: { maximum: 500 }
  validates :password, length: { in: 6..20 }
  validates :registration_number, length: { is: 6 }
end
```

Le possibili opzioni di limitazione della lunghezza sono:

- `:minimum` - L'attributo non può avere meno della lunghezza specificata.
- `:maximum` - L'attributo non può avere più della lunghezza specificata.
- `:in` (o `:within` ) - La lunghezza dell'attributo deve essere inclusa in un dato intervallo. Il valore per questa opzione deve essere un intervallo.
- `:is` - La lunghezza dell'attributo deve essere uguale al valore specificato.

## Convalida del raggruppamento

A volte è utile avere più convalida per utilizzare una condizione. Può essere facilmente raggiunto usando with_options.

```
class User < ApplicationRecord
  with_options if: :is_admin? do |admin|
    admin.validates :password, length: { minimum: 10 }
    admin.validates :email, presence: true
  end
end
```

Tutte le convalide all'interno del blocco with_options avranno automaticamente superato la condizione se:: is_admin?

## Convalide personalizzate

È possibile aggiungere le proprie convalide aggiungendo nuove classi ereditate da `ActiveModel::Validator` o da `ActiveModel::EachValidator` . Entrambi i metodi sono simili ma funzionano in modi leggermente diversi:

#### `ActiveModel::Validator` e `validates_with`

Implementa il metodo di `validate` che prende un record come argomento ed esegue la convalida su di esso. Quindi utilizzare `validates_with` con la classe sul modello.

```
# app/validators/starts_with_a_validator.rb
class StartsWithAValidator < ActiveModel::Validator
  def validate(record)
    unless record.name.starts_with? 'A'
      record.errors[:name] << 'Need a name starting with A please!'
    end
  end
end

class Person < ApplicationRecord
  validates_with StartsWithAValidator
end
```

#### `ActiveModel::EachValidator` e `validate`

Se si preferisce utilizzare il nuovo validatore utilizzando il metodo di `validate` comune su un singolo parametro, creare una classe che eredita da `ActiveModel::EachValidator` e implementare il metodo `validate_each` che accetta tre argomenti: `record` , `attribute` e `value` :

```
class EmailValidator < ActiveModel::EachValidator
  def validate_each(record, attribute, value)
    unless value =~ /\A([^@\s]+)@((?:[-a-z0-9]+\.)+[a-z]{2,})\z/i
      record.errors[attribute] << (options[:message] || 'is not an email')
    end
  end
end

class Person < ApplicationRecord
  validates :email, presence: true, email: true
end
```

Maggiori informazioni sulle guide di Rails .

### Convalida il formato di un attributo

Convalida che il valore di un attributo corrisponde a un'espressione regolare utilizzando il `format` e l'opzione `with` .

```
class User < ApplicationRecord
  validates :name, format: { with: /\A\w{6,10}\z/ }
end
```

Puoi anche definire una costante e impostarne il valore su un'espressione regolare e passarla all'opzione `with:` . Questo potrebbe essere più conveniente per espressioni regolari davvero complesse

```
PHONE_REGEX = /\A\(\d{3}\)\d{3}-\d{4}\z/
validates :phone, format: { with: PHONE_REGEX }
```

Il messaggio di errore predefinito `is invalid` . Questo può essere modificato con l'opzione `:message` .

```
validates :bio, format: { with: /\A\D+\z/, message: "Numbers are not allowed" }
```

Anche il contrario risponde e puoi specificare che un valore *non* deve corrispondere a un'espressione regolare con l'opzione `without:`

## Convalida l'inclusione di un attributo

Puoi verificare se un valore è incluso in un array usando l' `inclusion:` helper. L'opzione `:in` e il suo alias,: `:within` mostra l'insieme di valori accettabili.

```
class Country < ApplicationRecord
  validates :continent, inclusion: { in: %w(Africa Antartica Asia Australia
                                            Europe North America South America) }
end
```

Per verificare se un valore non è incluso in una matrice, utilizzare l' `exclusion:` helper

```
class User < ApplicationRecord
  validates :name, exclusion: { in: %w(admin administrator owner) }
end
```

## Convalida condizionale

A volte potrebbe essere necessario convalidare la registrazione solo in determinate condizioni.

```
class User < ApplicationRecord
  validates :name, presence: true, if: :admin?

  def admin?
    conditional here that returns boolean value
  end
end
```

Se il tuo condizionale è veramente piccolo, puoi usare un Proc:

```
class User < ApplicationRecord
  validates :first_name, presence: true, if: Proc.new { |user| user.last_name.blank? }
end
```

Per condizionale negativo è possibile utilizzare a `unless` :

```
class User < ApplicationRecord
  validates :first_name, presence: true, unless: Proc.new { |user| user.last_name.present? }
end
```

Puoi anche passare una stringa, che verrà eseguita tramite `instance_eval` :

```
class User < ApplicationRecord
  validates :first_name, presence: true, if: 'last_name.blank?'
end
```

## Conferma dell'attributo

Dovresti usare questo quando hai due campi di testo che dovrebbero ricevere esattamente lo stesso contenuto. Ad esempio, potresti voler confermare un indirizzo email o una password. Questa convalida crea un attributo ***virtuale il*** cui nome è il nome del campo che deve essere confermato con `_confirmation` aggiunto.

```
class Person < ApplicationRecord
  validates :email, confirmation: true
end
```

**Nota** Questo controllo viene eseguito solo se `email_confirmation` non è nullo.

Per richiedere la conferma, assicurati di aggiungere un controllo di presenza per l'attributo di conferma.

```
class Person < ApplicationRecord
  validates :email,      confirmation: true
  validates :email_confirmation, presence: true
end
```

fonte

## Utilizzo: su opzione

L'opzione `:on` consente di specificare quando deve avvenire la convalida. Il comportamento predefinito per tutti gli helper di convalida incorporati deve essere eseguito su save (sia quando si crea un nuovo record sia quando lo si aggiorna).

```
class Person < ApplicationRecord
  # it will be possible to update email with a duplicated value
  validates :email, uniqueness: true, on: :create

  # it will be possible to create the record with a non-numerical age
  validates :age, numericality: true, on: :update

  # the default (validates on both create and update)
  validates :name, presence: true
end
```

Leggi Validazioni ActiveRecord online: https://riptutorial.com/it/ruby-on-rails/topic/2105/validazioni-activerecord

# Capitolo 74: Visualizzazioni

## Examples

### parziali

I modelli parziali (partial) sono un modo per rompere il processo di rendering in blocchi più gestibili. I partial consentono di estrarre parti di codice dai modelli per separare i file e riutilizzarli anche nei modelli.

Per *creare* un partial, crea un nuovo file che inizia con un carattere di sottolineatura: `_form.html.erb`

Per *rendere* un partial come parte di una vista, usa il metodo render nella vista: `<%= render "form" %>`

- Nota, il carattere di sottolineatura viene omesso durante il rendering
- Un partial deve essere reso utilizzando il suo percorso se si trova in una cartella diversa

Per *passare* una variabile nel partial come variabile locale, usa questa notazione:

```
<%= render :partial => 'form', locals: { post: @post } %>
```

I partial sono utili anche quando è necessario *riutilizzare* esattamente lo stesso codice ( **filosofia DRY** ).

Ad esempio, per riutilizzare il codice `<head>` , creare un partial denominato `_html_header.html.erb` , inserire il codice `<head>` da riutilizzare e rendere il partial quando necessario: `<%= render 'html_header' %>` .

# Oggetti parziali

Anche gli oggetti che rispondono a `to_partial_path` possono essere visualizzati, come in `<%= render @post %>` . Di default, per i modelli ActiveRecord, questo sarà qualcosa di simile a `posts/post` , quindi in realtà il rendering di `@post` , il file `views/posts/_post.html.erb` sarà reso.

Un `post` denominato locale verrà assegnato automaticamente. Alla fine, `<%= render @post %>` è una mano breve per `<%= render 'posts/post', post: @post %>` .

Possono anche essere fornite raccolte di oggetti che rispondono a `to_partial_path` , come `<%= render @posts %>` . Ogni oggetto sarà reso consecutivamente.

# Global Partials

Per creare un partial globale che può essere utilizzato ovunque senza fare riferimento al suo

percorso esatto, il partial deve essere posizionato nel percorso `views/application` . L'esempio precedente è stato modificato di seguito per illustrare questa funzionalità.

Ad esempio, questo è un percorso per `app/views/application/_html_header.html.erb:` parziale `app/views/application/_html_header.html.erb:`

Per rendere questo parziale globale ovunque, usa `<%= render 'html_header' %>`

### AssetTagHelper

Per consentire ai binari di collegare automaticamente e correttamente le risorse (css / js / images) nella maggior parte dei casi, si desidera utilizzare gli helper integrati. ( Documentazione ufficiale )

# Aiutanti di immagini

## *percorso_immagine*

Ciò restituisce il percorso di un asset immagine in `app/assets/images` .

```
image_path("edit.png") # => /assets/edit.png
```

## *URL dell'immagine*

Ciò restituisce l'URL completo a una risorsa immagine in `app/assets/images` .

```
image_url("edit.png") # => http://www.example.com/assets/edit.png
```

## *image_tag*

Usa questo helper se vuoi includere un `<img src="" />` -tag con il set sorgente.

```
image_tag("icon.png") # => <img src="/assets/icon.png" alt="Icon" />
```

# Helper JavaScript

## *javascript_include_tag*

Se si desidera includere un file JavaScript nella propria vista.

```
javascript_include_tag "application" # => <script src="/assets/application.js"></script>
```

## *javascript_path*

Questo restituisce il percorso del tuo file JavaScript.

```
javascript_path "application" # => /assets/application.js
```

### *javascript_url*

Questo restituisce l'URL completo del tuo file JavaScript.

```
javascript_url "application" # => http://www.example.com/assets/application.js
```

## Aiutanti foglio di stile

### *stylesheet_link_tag*

Se si desidera includere un file CSS nella propria vista.

```
stylesheet_link_tag "application" # => <link href="/assets/application.css" media="screen"
rel="stylesheet" />
```

### *stylesheet_path*

Ciò restituisce il percorso dell'asset del foglio di stile.

```
stylesheet_path "application" # => /assets/application.css
```

### *stylesheet_url*

Questo restituisce l'URL completo della risorsa del foglio di stile.

```
stylesheet_url "application" # => http://www.example.com/assets/application.css
```

## Esempio di utilizzo

Quando crei una nuova app per rails, avrai automaticamente due di questi helper in `app/views/layouts/application.html.erb`

```
<%= stylesheet_link_tag    'application', media: 'all', 'data-turbolinks-track': 'reload' %>
<%= javascript_include_tag 'application', 'data-turbolinks-track': 'reload' %>
```

Questo produce:

```
// CSS
```

```
<link rel="stylesheet" media="all" href="/assets/application.self-
e19d4b856cacba4f6fb0e5aa82a1ba9aa4ad616f0213a1259650b281d9cf6b20.css?body=1" data-turbolinks-
track="reload" />
// JavaScript
<script src="/assets/application.self-
619d9bf310b8eb258c67de7af745cafbf2a98f6d4c7bb6db8e1b00aed89eb0b1.js?body=1" data-turbolinks-
track="reload"></script>
```

## Struttura

Come Rails segue il pattern M **V** C Le `Views` sono dove i tuoi "templates" sono per le tue azioni.

Supponiamo che tu abbia un controller `articles_controller.rb` . Per questo controller avresti una cartella in viste chiamata `app/views/articles` :

```
app
|-- controllers
|   '-- articles_controller.rb
|
'-- views
    '-- articles
    |   |- index.html.erb
    |   |- edit.html.erb
    |   |- show.html.erb
    |   |- new.html.erb
    |   '- _partial_view.html.erb
    |
    '-- [...]
```

Questa struttura ti consente di avere una cartella per ogni controller. Quando si chiama un'azione nel controller, la vista appropriata verrà resa automaticamente.

```
// articles_controller.rb
class ArticlesController < ActionController::Base
  def show
  end
end

// show.html.erb
<h1>My show view</h1>
```

## Sostituisci il codice HTML in Views

Se hai mai voluto determinare il contenuto html da stampare su una pagina durante il runtime, allora rails ha un'ottima soluzione per questo. Ha qualcosa chiamato **content_for** che ci consente di passare un blocco a una vista di binari. Si prega di controllare l'esempio di seguito,

**Dichiara content_for**

```
<div>
  <%= yield :header %>
</div>
```

```
<% content_for :header do %>
 <ul>
   <li>Line Item 1</li>
   <li>Line Item 2</li>
 </ul>
<% end %>
```

## HAML: un modo alternativo di utilizzare nelle tue visualizzazioni

HAML (linguaggio di markup di astrazione HTML) è un modo bello ed elegante per descrivere e progettare l'HTML delle tue visualizzazioni. Invece di aprire e chiudere i tag, HAML utilizza il rientro per la struttura delle tue pagine. Fondamentalmente, se qualcosa deve essere posizionato all'interno di un altro elemento, lo si indenta semplicemente usando un punto di tabulazione. Le schede e lo spazio bianco sono importanti in HAML, quindi assicurati di utilizzare sempre la stessa quantità di schede.

**Esempi:**

```
#myview.html.erb
<h1><%= @the_title %></h1>
<p>This is my form</p>
<%= render "form" %>
```

E in HAML:

```
#myview.html.haml
%h1= @the_title
%p
    This is my form
= render 'form'
```

Vedete, la struttura del layout è molto più chiara rispetto all'utilizzo di HTML e ERB.

**Installazione**

Basta installare la gemma usando

```
gem install haml
```

e aggiungi la gemma al Gemfile

```
gem "haml"
```

Per utilizzare HAML anziché HTML / ERB, basta sostituire le estensioni dei file delle tue visualizzazioni da `something.html.erb` a `something.html.haml` .

**Consigli rapidi**

Elementi comuni come le div possono essere scritti in modo breve

HTML

```
<div class="myclass">My Text</div>
```

HAML

```
%div.myclass
```

HAML, stenografia

```
.myclass
```

**attributi**

HTML

```
<p class="myclass" id="myid">My paragraph</p>
```

HAML

```
%p{:class => "myclass", :id => "myid"} My paragraph
```

**Inserimento di codice rubino**

Puoi inserire il codice rubino con i segni = e -.

```
= link_to "Home", home_path
```

Il codice che inizia con **=** sarà eseguito e incorporato nel documento.

Il codice che inizia con **-** verrà eseguito, ma non inserito nel documento.

**Documentazione completa**

HAML è molto facile da iniziare, ma è anche molto complesso, quindi consiglierò di leggere la documentazione .

Leggi Visualizzazioni online: https://riptutorial.com/it/ruby-on-rails/topic/850/visualizzazioni

# Titoli di coda

| S. No | Capitoli | Contributors |
|---|---|---|
| 1 | Iniziare con Ruby on Rails | Abhishek Jain, Adam Lassek, Ajay Barot, animuson, ArtOfCode, Aswathy, Community, Darpan Chhatravala, Darshan Patel, Deepak Mahakale, fybw id, Geoffroy, hschin, hvenables, Jon Wood, kfrz, Kirti Thorat, Lorenzo Baracchi, Luka Kerr, MauroPorrasP, michaelpri, nifCody, Niyanta, olive_tree, RADan, RareFever, Richard Hamilton, sa77, saadlulu, sahil, Sathishkumar Jayaraj, Simone Carletti, Stanislav Valášek, theoretisch, tpei, Undo, uzaif, Yana |
| 2 | ActionCable | Ich, Sladey, Undo |
| 3 | ActionController | Adam Lassek, Atul Khanduri, Deep, Fire-Dragon-DoL, Francesco Lupo Renzi, jackerman09, RamenChef, Sven Reuter |
| 4 | ActionMailer | Adam Lassek, Atul Khanduri, jackerman09, owahab, Phil Ross, Richard Hamilton, Rodrigo Argumedo, William Romero |
| 5 | ActiveJob | Brian, owahab |
| 6 | ActiveModel | Adam Lassek, RamenChef |
| 7 | ActiveRecord | Adam Lassek, AnoE, Bijal Gajjar, br3nt, D-side, Francesco Lupo Renzi, glapworth, jeffdill2, Joel Drapper, Luka Kerr, maartenvanvliet, marcamillion, Mario Uher, powerup7, Sebastialonso, Simone Carletti, Sven Reuter, walid |
| 8 | ActiveRecord Associations | giniouxe, Hardik Upadhyay, Khanh Pham, Luka Kerr, Manish Agarwal, Niyanta, RareFever, Raynor Kuang, Sapna Jindal |
| 9 | ActiveSupport | Adam Lassek |
| 10 | Aggiornamento di Rails | hschin, michaelpri, Rodrigo Argumedo |
| 11 | Aggiungi il pannello di amministrazione | Ahsan Mahmood, MSathieu |
| 12 | Aggiunta di un Amazon RDS all'applicazione rails | Sathishkumar Jayaraj |

| | | |
|---|---|---|
| 13 | API Rails | Adam Lassek, hschin |
| 14 | Asset Pipeline | fybw id, Robin |
| 15 | Autenticate Api usando Devise | Vishal Taj PM |
| 16 | Autenticazione API Rails 5 | HParker |
| 17 | Autenticazione utente in Rails | Abhinay, Ahsan Mahmood, Antarr Byrd, ArtOfCode, dgilperez, Kieran Andrews, Luka Kerr, Qchmqs, uzaif, |
| 18 | Autorizzazione con CanCan | 4444, Ahsan Mahmood, dgilperez, mlabarca, toobulkeh |
| 19 | Blocco ActiveRecord | Adam Lassek, fatfrog, Muaaz Rafi |
| 20 | caching | ArtOfCode, Cuisine Hacker, Khanh Pham, RamenChef, tirdadc |
| 21 | Colonne ActiveRecord multiuso | Fabio Ros |
| 22 | Configura Angolare con Rails | B8vrede, Rory O'Kane, Umar Khan |
| 23 | Configurazione | Ali MasudianPour, Undo |
| 24 | Constantize costante | Eric Bouchut, Ryan K |
| 25 | Convenzioni di denominazione | Andrey Deineko, Atul Khanduri, br3nt, Flambino, giniouxe, hgsongra, Luka Kerr, Marko Kacanski, Muhammad Abdullah, Sven Reuter, Xinyang Li |
| 26 | Debug | Adam Lassek, Dénes Papp, Dharam, Kelseydh, sa77, titan |
| 27 | Distribuzione di un'app Rails su Heroku | B Liu, hschin |
| 28 | elasticsearch | Don Giovanni, Luc Boissaye |
| 29 | Ereditarietà di una tabella singola | Niyanta, Ruslan, Slava.K, toobulkeh, Vishal Taj PM |
| 30 | Formare aiutanti | aisflat439, owahab, Richard Hamilton, Simon Tsang, Slava.K |
| 31 | Funzione di pagamento in binari | ppascualv, Sathishkumar Jayaraj |

| | | |
|---|---|---|
| 32 | Gems | Deep, hschin, ma_il, MMachinegun, RamenChef |
| 33 | I18n - Internazionalizzazione | Cyril Duchon-Doris, Francesco Lupo Renzi, Frederik Spang, gwcodes, Jorge Najera T, Lahiru, RamenChef |
| 34 | ID amichevole | Thang Le Sy |
| 35 | Importa interi file CSV da una cartella specifica | fool |
| 36 | Integrazione di React.js con Rails mediante Hyperloop | Mitch VanDuyn |
| 37 | Interfaccia di query ActiveRecord | Adam Lassek, Ajay Barot, Avdept, br3nt, dnsh, Fabio Ros, Francesco Lupo Renzi, giniouxe, jeffdill2, MikeAndr, Muhammad Abdullah, Niyanta, powerup7, rdnewman, Reboot, Robin, sa77, Vishal Taj PM |
| 38 | Lavori attivi | tirdadc |
| 39 | Logger rotaie | Alejandro Montilla, hgsongra |
| 40 | Memorizzazione sicura delle chiavi di autenticazione | DawnPaladin |
| 41 | Migrazioni di ActiveRecord | Adam Lassek, Aigars Cibuļskis, Alex Kitchens, buren, Deepak Mahakale, Dharam, DSimon, Francesco Lupo Renzi, giniouxe, Hardik Kanjariya ツ, hschin, jeffdill2, Kirti Thorat, KULKING, maartenvanvliet, Manish Agarwal, Milo P, Mohamad, MZaragoza, nomatteus, Reboot, Richard Hamilton, rii, Robin, Rodrigo Argumedo, rony36, Rory O'Kane, tessi, uzaif, webster |
| 42 | Modifica il fuso orario predefinito | Mihai-Andrei Dinculescu |
| 43 | Modificare un ambiente dell'applicazione Rails predefinito | Whitecat |
| 44 | Modulo annidato in Ruby on Rails | Arslan Ali |
| 45 | Mongoid | Ryan K, tes |
| 46 | Motivo decorativo | Adam Lassek |

| | | |
|---|---|---|
| 47 | Operaia | Rafael Costa |
| 48 | Organizzazione di classe | Deep, hadees, HParker |
| 49 | Parole riservate | Emre Kurt |
| 50 | PDF di gamberi | Awais Shafqat |
| 51 | Percorso superficiale | Darpan Chhatravala |
| 52 | Quadri rotaie nel corso degli anni | Shivasubramanian A |
| 53 | Rails 5 | thiago araujo |
| 54 | Rails Cookbook - Advanced Rails: ricette / apprendimento e tecniche di codifica | Milind |
| 55 | Rails Engine - Modular Rails | Mayur Shah |
| 56 | Rails -Engines | Deepak Kabbur |
| 57 | Rails genera comandi | Adam Lassek, ann, Deepak Mahakale, Dharam, Hardik Upadhyay, jackerman09, Jeremy Green, marcamillion, Milind, Muhammad Abdullah, nomatteus, powerup7, Reub, Richard Hamilton |
| 58 | Reagire con Rails usando gemme react-rails | Kimmo Hintikka, tirdadc |
| 59 | Rota Best Practice | Adam Lassek, Brandon Williams, Gaston, giniouxe, Hardik Upadhyay, inye, Joel Drapper, Josh Caswell, Luka Kerr, ma_il, msohng, Muaaz Rafi, piton4eg, powerup7, rony36, Sri, Tom Lazar |
| 60 | Rotaie sulla finestra mobile | ppascualv, Sathishkumar Jayaraj |
| 61 | Routing | Adam Lassek, advishnuprasad, Ahsan Mahmood, Alejandro Babio, Andy Gauge, AppleDash, ArtOfCode, Baldrick, cl3m, Cyril Duchon-Doris, Deepak Mahakale, Dharam, Eliot Sykes, esthervillars, Fabio Ros, Fire-Dragon-DoL, Francesco Lupo Renzi, giniouxe, Giuseppe, Hassan Akram, Hizqeel, HungryCoder, jkdev, John Slegers, Jon Wood, Kevin Sylvestre, |

| | | |
|---|---|---|
| | | Kieran Andrews, Kirti Thorat, KULKING, Leito, Mario Uher, Milind, Muhammad Faisal Iqbal, niklashultstrom, nuclearpidgeon, pastullo, Rahul Singh, rap-2-h, Raynor Kuang, Richard Hamilton, Robin, rogerdpack, Rory O'Kane, Ryan Hilbert, Ryan K, Silviu Simeria, Simone Carletti, sohail khalil, Stephen Leppik, TheChamp, Thor odinson, Undo, Zoran, |
| 62 | RSpec e Ruby on Rails | Ashish Bista, Scott Matthewman, Simone Carletti |
| 63 | Serializzatori di modelli attivi | Flip, owahab |
| 64 | Stati modello: AASM | Lomefin |
| 65 | Strumenti per l'ottimizzazione e la pulizia del codice Ruby on Rails | Akshay Borade |
| 66 | Test delle applicazioni Rails | HParker |
| 67 | Transazioni ActiveRecord | abhas, Adam Lassek |
| 68 | Turbolinks | Mark |
| 69 | Upload di file | Sergey Khmelevskoy |
| 70 | Utilizzo di Google Maps con Rails | fiedl |
| 71 | Validazioni ActiveRecord | Adam Lassek, Colin Herzog, Deepak Mahakale, dgilperez, dodo121, giniouxe, Hai Pandu, Hardik Upadhyay, mmichael, Muhammad Abdullah, pablofullana, Richard Hamilton |
| 72 | Visualizzazioni | danirod, dgilperez, elasticman, Luka Kerr, MikeC, MMachinegun, Pragash, RareFever |