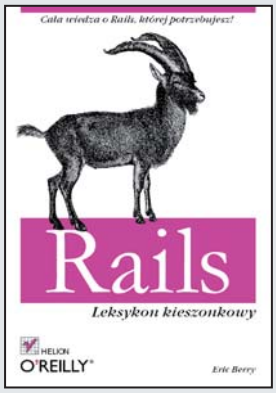
Rails. Leksykon kieszonkowy



Autor: Eric Berry
Tłumaczenie: Anna Trojan
ISBN: 978-83-246-2131-6
Tytuł oryginału: COM+ Developer's Guide
Format: 115x170, stron: 224

Cała wiedza o Rails, której potrzebujesz!

- Jak zainstalować i skonfigurować Rails?
- Jak wykorzystać możliwości technologii ActiveRecord?
- Jak stworzyć dynamiczną aplikację, korzystając z technologii AJAX?

Ruby on Rails przebojem wdarł się na rynek szkieletów aplikacji internetowych. Stworzony w architekturze MVC, został niezmiernie dobrze przyjęty przez programistów z całego świata. Niewątpliwie wpływ na ten fakt miały założenia poczynione przez autora projektu – łatwość i przyjemność tworzenia kodu. Powszechnie wiadomo, że te atuty mają niebagatelny wpływ na szybkość tworzenia aplikacji internetowych. Dzięki popularności Ruby on Rails również znajomość języka Ruby, pochodzącego przecież z Dalekiego Wschodu, staje się powszechniejsza.

W tej niezwykłej książce znajdziesz odpowiedzi na wszystkie pytania, które mogą pojawić się w trakcie kodowania nowej aplikacji. Dowiesz się, jak skonfigurować Rails, testować napisany kod, wykorzystać technologię ActiveRecord oraz przygotować widoki. Ponadto zdobędziesz wiedzę na temat używania technologii AJAX i REST, nauczysz się wykorzystywać usługi sieciowe oraz logować ważne informacje w trakcie pracy Twojej aplikacji. Z wiedzy tu zawartej możesz skorzystać szybko i w każdej chwili, a co najważniejsze, nie będziesz musiał przedzierać się przez setki niepotrzebnych stron. Tu znajdziesz tylko wiedzę niezbędną w Twojej pracy!

- Instalacja Rails
- Struktura plików
- Konfiguracja Rails
- Wykorzystanie skryptów
- Konfiguracja środowiska
- Sposoby przeprowadzania testów w Rails
- Używanie technologii ActiveRecord
- Zarządzanie danymi w bazie
- Zastosowanie ActionController
- Tworzenie widoków
- Sposoby użycia technologii AJAX i REST w Rails
- Wykorzystanie usług sieciowych
- Logowanie informacji w trakcie pracy aplikacji
- Stosowanie metod pomocniczych

Wykorzystaj Rails w Twoim projekcie szybko i przyjemnie!

Helion SA
ul. Kościuszki 1c
44-100 Gliwice
tel. 032 230 98 63
e-mail: helion@helion.pl

Spis treści

Polecenia oraz konfiguracja Rails

Po zainstalowaniu Rails można użyć polecenia `rails` do generowania nowych aplikacji Rails z domyślną strukturą katalogów oraz konfiguracją dla określonej ścieżki.

By utworzyć aplikację Rails o nazwie *myapp*, należy wpisać:

```
rails myapp
```

Po wykonaniu tego polecenia zobaczymy listę katalogów oraz plików wygenerowanych przez nie. To nasza aplikacja Rails; *myapp* będzie folderem głównym, inaczej RAILS ROOT.

Użycie oraz opcje

Pomoc dla polecenia `rails` można uzyskać za pomocą:

```
rails --help
```

Użycie

```
rails [/ścieżka/do/aplikacji][opcje]
```

Opcje

`-r, --ruby=`*`ścieżka`*
: Ścieżka do wybranych plików binarnych języka Ruby.

`-d, --database=`*`nazwa`*
: Konfiguracja dla określonej bazy danych (na przykład `mysql`, `oracle`, `postgresql`, `sqlite2`, `sqlite3`).

`-f, --freeze`
: Zamrożenie Rails w katalogu *vendor/rails* z gemów generujących szkielet.

`-v, --version`
: Pokazuje numer wersji Rails i kończy działanie.

`-p, --pretend`
: Jest wykonywane, ale nie wprowadza żadnych zmian.

`--force`
: Nadpisuje istniejące pliki.

`-s, --skip`
: Pomija istniejące pliki.

`-q, --quiet`
: Blokuje wyświetlanie normalnych danych wyjściowych.

`-t, --backtrace`
: Debugowanie; w przypadku błędów pokazuje ślad wykonanych czynności.

`-c, --svn`
: Modyfikuje pliki z użyciem Subversion (`svn` musi być w ścieżce).

Struktura plików Rails

Po wygenerowaniu aplikacji Rails utworzone zostają domyślny katalog oraz struktura plików (tabela 1.1).

Tabela 1.1. Struktura plików aplikacji Rails

Ścieżka	Opis
app	Przechowuje kod specyficzny dla określonej aplikacji.
app/controllers	Przechowuje kontrolery, które dla automatycznego odwzorowania adresów URL powinny mieć nazwy takie jak *user_controller.rb*. Wszystkie kontrolery powinny pochodzić od `ApplicationController`, który z kolei pochodzi od `ActionController::Base`.
app/models	Przechowuje modele, które powinny nosić nazwy takie jak *product.rb*. Większość modeli pochodzi od `ActiveRecord::Base`.
app/views	Przechowuje pliki szablonów dla widoków. Powinny one nosić ustandaryzowane nazwy, takie jak *users/index.html.erb* w przypadku akcji `UsersController index`. Wszystkie widoki wykorzystują składnię eRuby.

Tabela 1.1. Struktura plików aplikacji Rails — ciąg dalszy

Ścieżka	Opis
app/views/layouts	Przechowuje pliki szablonów dla układów dokumentów, jakie mają być wykorzystane z widokami. Przypomina to metodę wspólnego nagłówka czy stopki opakowujących widoki. W widokach definiuje się układ dokumentu za pomocą układu `:default` i tworzy plik o nazwie *default.html.erb*. Wewnątrz *default.html.erb* wywołuje się `<%= yield %>` w celu wygenerowania widoku z użyciem określonego układu dokumentu.
app/helpers	Przechowuje metody pomocnicze, które powinny nosić nazwy takie jak *users_helper.rb*. Są one generowane automatycznie, kiedy dla kontrolerów wykorzystuje się polecenie `script/generate`. Metody pomocnicze można wykorzystać do opakowania funkcjonalności przeznaczonej dla widoków w metody.
config	Pliki konfiguracyjne dla środowiska Rails, mapy tras, bazy danych oraz innych zależności.
db	Zawiera schemat bazy danych znajdujący się w *schema.rb.db/migrate*. Zawiera wszystkie sekwencje migracji dla schematu.
doc	W tym katalogu przechowywana będzie dokumentacja aplikacji po jej wygenerowaniu za pomocą `rake doc:app`.
lib	Biblioteki specyficzne dla aplikacji. Tak naprawdę każdy rodzaj własnego kodu, który nie jest kontrolerem, modelem ani metodą pomocniczą. Katalog ten znajduje się w ścieżce ładowania.
public	Katalog dostępny dla serwera WWW. Zawiera podkatalogi przeznaczone dla obrazków, arkuszy stylów oraz skryptów JavaScript. Zawiera również dyspozytor (ang. *dispatcher*) i domyślne pliki HTML. Powinien być ustawiony jako DOCUMENT ROOT serwera WWW.
script	Skrypty pomocnicze służące do automatyzacji oraz generacji.
test	Testy jednostkowe oraz funkcjonalne wraz z fiksturami. Kiedy wykorzystuje się skrypty `script/generate`, pliki szablonów testów zostają dla nas wygenerowane i umieszczone w tym katalogu.
vendor	Biblioteki zewnętrzne, od których uzależniona jest nasza aplikacja. Zawiera również podkatalog *plugins*. Katalog ten znajduje się w ścieżce ładowania.

Konfiguracja Rails

Każda aplikacja Rails oparta jest na plikach konfiguracyjnych określających jej sposób działania. Pliki te znajdują się w folderze *config*:

boot.rb
: Ładuje plik programu rozruchowego. Zazwyczaj plik ten nie musi być modyfikowany.

routes.rb
: Plik konfiguracyjny określający trasy.

environment.rb
: Ogólne ustawienia konfiguracyjne dla aplikacji Rails.

environments/development.rb
: Konfiguracja specyficzna dla środowiska programistycznego.

environments/test.rb
: Konfiguracja specyficzna dla środowiska testowego.

environments/production.rb
: Konfiguracja specyficzna dla środowiska produkcyjnego.

database.yml
: Konfiguracja połączenia z bazą danych.

initializers/inflections.rb
: Dodaje nowe reguły fleksyjne do aplikacji (na przykład dotyczące liczby mnogiej, pojedynczej, rzeczowników niepoliczalnych).

initializers/mime_types.rb
: Dodaje nowe typy MIME do użycia w blokach `respond_to` (na przykład rtf lub iPhone).

initializers/new_rails_defaults.rb
: Te ustawienia zmieniają zachowanie aplikacji Rails 2 i będą domyślne dla Rails 3. Plik ten w Rails 3 stanie się przestarzały.

Więcej informacji na temat konfiguracji tras, środowisk oraz baz danych można znaleźć w odpowiednich częściach niniejszej książki.

Skrypty

Rails zawiera skrypty pomocnicze służące do automatyzacji oraz generowania kodu. Wykorzystując te skrypty, programiści mogą szybko budować aplikacje, zachowując jednocześnie kontrolę nad wygenerowaną zawartością.

script/about

Wyświetla informacje o środowisku aplikacji:

```
Ruby version            1.8.6 (universal-darwin9.0)
RubyGems version        1.1.1
Rails version           2.1.0
Active Record version   2.1.0
Action Pack version     2.1.0
Active Resource version 2.1.0
Action Mailer version   2.1.0
Active Support version  2.1.0
Application root        /Users/berry/Sites/myapp
Environment             development
Database adapter        sqlite3
Database schema version 2
```

script/console

Konsola daje nam dostęp do środowiska Rails, w którym można wykonywać interakcje z modelem domeny. Wszystkie części aplikacji są tutaj skonfigurowane w taki sposób, jakby aplikacja działała. Można badać modele domeny, zmieniać wartości i zapisywać do bazy danych.

Konsolę można uruchomić za pomocą poniższego skryptu:

```
./script/console
```

Uruchomienie konsoli bez argumentów spowoduje jej działanie z wykorzystaniem środowiska programistycznego.

By zatrzymać konsolę, należy wpisać:

```
quit lub exit
```

W późniejszych wersjach Rails można przeładowywać modele oraz kontrolery za pomocą poniższego polecenia:

```
reload!
```

By przywrócić aplikację do pierwotnych wartości, należy wpisać:

```
Dispatcher.reset_application!
```

Użycie

```
./script/console [środowisko] [opcje]
```

Opcje

`-s, --sandbox`
: Przy wyjściu przywraca modyfikacje bazy danych do stanu początkowego.

`--irb=[irb]`
: Uruchamia inny *irb*.

Wskazówka

Wykorzystywanie konsoli w trybie `sandbox` daje programiście Rails ogromne możliwości. Jeśli na przykład utworzymy skrypt uaktualniający wszystkie wiersze tabeli, możemy najpierw uruchomić ten skrypt testowo, w trybie `sandbox`, w oparciu o istniejące dane, a później sprawdzić, czy zmiany zostały przeprowadzone w poprawny sposób. Jeśli coś pójdzie nie tak, wystarczy wyjść z konsoli, a w bazie danych nie zostaną wprowadzone żadne zmiany.

---

script/destroy

Skrypt ten zniszczy wszystkie pliki utworzone przez odpowiadające mu polecenie `script/generate`. Przykładowo polecenie `script/destroy migration CreatePost` usunie odpowiedni plik ###_*create_post.rb* znajdujący się w katalogu *db/migrate*, natomiast polecenie `script/destroy scaffold Post` usunie kontroler oraz widoki dla `Post` wraz z modelem i migracją oraz wszystkimi powiązanymi testami, a także wiersz `map.resources :post` w pliku *config/routes.rb*.

Użycie

```
./script/destroy generator [opcje] [argumenty]
```

Przykład

```
./script/destroy controller Products

      rm app/helpers/products_helper.rb
      rm test/functional/products_controller_test.rb
      rm app/controllers/products_controller.rb
   rmdir test/functional
notempty test
   rmdir app/views/products
notempty app/views
notempty app
notempty app/helpers
notempty app
notempty app/controllers
notempty app
```

script/generate

Generatory wykorzystywane są do tworzenia kodu służącego do natychmiastowego użycia w aplikacji Rails. Po uruchomieniu generatora (za pomocą `script/generate`) nowe pliki (kontrolery, modele, widoki) są generowane i dodawane do aplikacji. Możliwe

jest również tworzenie własnych generatorów. Więcej informacji na temat własnych generatorów można znaleźć w podcaście Ryana Batesa znajdującym się pod adresem *http://railscasts.com/episodes/58*.

By otrzymać pomoc dotyczącą skryptu generatora, należy wpisać:

```
./script/generate --help
```

Pomoc na temat konkretnego generatora można uzyskać, wpisując:

```
./script/generate [generator] --help
```

Rails zawiera kilka wbudowanych generatorów.

controller

Tworzy nowy kontroler wraz z widokami. Jako argumenty należy przekazać nazwę kontrolera, albo w pisowni wielbłądziej, albo ze znakiem _, a także listę widoków.

By utworzyć nowy kontroler wewnątrz modułu, należy podać nazwę kontrolera w postaci ścieżki, na przykład *nazwa_modułu/nazwa_kontrolera*.

Można również tworzyć podkontrolery dziedziczące po kontrolerze nadrzędnym, podając nazwę kontrolera w postaci `NazwaKontrolera ↪Nadrzędnego::NazwaKontroleraPodrzędnego`. Jeśli na przykład mamy kontroler o nazwie *admin* i chcemy utworzyć podkontroler o nazwie *users* dziedziczący filtr uwierzytelniający po kontrolerze nadrzędnym `AdminController`, możemy wpisać:

```
./script/generate controller Admin::Users
```

Powyższy kod generuje klasę kontrolera w *app/controllers/admin*, szablony widoku w *app/views/admin/controller_name*, klasę pomocniczą w *app/helpers/admin* i zbiór testów funkcjonalnych w *test/functional/admin*.

By kontroler potomny `UsersController` dziedziczył po `Admin ↪Controller`, będziemy musieli dodać poprawne dziedziczenie w pliku *app/controllers/admin/users_controller.rb*.

```
class UsersController < AdminController
```

Użycie

```
./script/generate controller NazwaKontrolera [opcje]
```

Przykład

```
./script/generate controller Product name:string
price:decimal quantity:integer
```

integration_test

Tworzy nowy test integracyjny. Jako argument należy przekazać nazwę testu, albo w pisowni wielbłądziej, albo ze znakiem _. Nowa klasa testowa zostanie wygenerowana w *test/integration/nazwa testu_test.rb*.

Użycie

```
./script/generate integration_test NazwaTestuIntegracyjnego
↪[opcje]
```

Przykład

```
./script/generate integration_test GeneralStories
```

mailer

Tworzy nowy program pocztowy (mailer) wraz z widokami. Jako argumenty należy przekazać nazwę programu pocztowego, albo w pisowni wielbłądziej, albo ze znakiem _, oraz opcjonalną listę e-maili jako argumenty.

Powoduje to wygenerowanie klasy programu pocztowego w *app/models*, szablonów widoków w *app/views/nazwa_programu*, testów jednostkowych w *test/unit* oraz fikstur w *test/fixtures*.

Użycie

```
./script/generate mailer NazwaProgramuPocztowego [opcje]
```

Przykład

```
./script/generate mailer Notifications signup
↪forgot_password invoice
```

migration

Tworzy nową migrację bazy danych. Jako argumenty należy przekazać nazwę migracji, albo w pisowni wielbłądziej, albo ze znakiem _, oraz opcjonalną listę par atrybutów.

Klasa migracji generowana jest w *db/migrate* i poprzedzona jest datą.

Migrację można nazwać w dowolny sposób. Preferowanym standardem jest jednak nazwa szczegółowo opisująca cel skryptu, na przykład `AddColumnsToTable` czy `RemoveColumnsFromTable`.

Użycie

```
./script/generate migration NazwaMigracji [opcje]
```

Przykład

```
./script/generate migration AddLastLoginToUsers
↪last_login:datetime
```

model

Tworzy nowy model, generując nową klasę w *app/models*, nową klasę migracji w *db/migrate*, test jednostkowy w *test/unit* oraz fikturę w *test/fixtures*. Jako argumenty należy przekazać nazwę modelu, albo w pisowni wielbłądziej, albo ze znakiem _, oraz opcjonalną listę par atrybutów.

Użycie

```
./script/generate model NazwaModelu [pole:typ, pole:typ]
```

Przykład

```
./script/generate model Book title:string description:text
↪pages:integer
```

observer

Tworzy nową klasę `Observer`. `Observer` umożliwia monitorowanie zdarzeń cyklu życia obiektu modelu poza samym modelem, a także pozwala na uniknięcie obciążenia klasy modelu logiką niestanowiącą jej jądra. Więcej informacji na temat tej klasy można znaleźć pod adresem *http://api.rubyonrails.org/classes/ActiveRecord/Observer.html*.

By wygenerować nowy obiekt `Observer`, należy jako argument przekazać jego nazwę, albo w pisowni wielbłądziej, albo ze znakiem _.

Generator tworzy klasę `Observer` w katalogu *app/models* oraz test jednostkowy w katalogu *test/unit*.

Użycie

```
./script/generate observer NazwaKlasyObserver [opcje]
```

Przykład

```
./script/generate observer Account
```

plugin

Tworzy nowy dodatek (plugin). Jako argument należy przekazać nazwę dodatku, albo w pisowni wielbłądziej, albo ze znakiem _. By dodać również generator przykładów, należy przekazać opcję `--with-generator`.

Tworzy dodatek w katalogu *vendor/plugins* wraz z plikiem *init.rb* oraz *README* i standardowe katalogi *lib*, *task* oraz *test*.

Użycie

```
./script/generate plugin NazwaDodatku [opcje]
```

Przykład

```
./script/generate plugin SimpleLayout
```

resource

Tworzy nowy zasób wraz z pustym modelem oraz kontrolerem przystosowanym do potrzeb aplikacji REST zorientowanej na zasoby. Należy przekazać pojedynczą nazwę modelu, albo w pisowni wielbłądziej, albo ze znakiem _, jako pierwszy argument oraz opcjonalną listę par atrybutów.

Użycie

```
./script/generate resource NazwaModelu [pole:typ, pole:typ]
```

Przykład

```
./script/generate resource Post title:string body:text
↪published:boolean
```

scaffold

Tworzy rusztowanie (ang. *scaffold*) całego zasobu, od modelu oraz migracji po kontroler i widoki, wraz z pełnym zestawem testów, i dodaje zasób do pliku *config/routes.rb*. Zasób jest gotowy do użycia jako punkt wyjścia dla aplikacji REST zorientowanej na zasoby.

Użycie

```
./script/generate scaffold NazwaModelu [pole:typ, pole:typ]
```

Przykład

```
./script/generate scaffold Comment user_id:integer body:text
```

session_migration

Tworzy migrację dodającą tabelę sesji wykorzystywaną przez magazyn sesji ActiveRecord. Jako argument należy przekazać nazwę migracji, albo w pisowni wielbłądziej, albo ze znakiem _.

Użycie

```
./script/generate session_migration NazwaMigracjiSesji [opcje]
```

Przykład

```
./script/generate session_migration CreateSessionTable
```

Więcej informacji na temat różnych typów sesji można znaleźć w podrozdziale „Sesje" w dalszej części książki.

script/performance

Rails zawiera kilka skryptów poprawiających wydajność aplikacji.

benchmarker

Testuje kilka razy wydajność jednej lub większej liczby instrukcji w środowisku Rails.

Użycie

```
./script/performance/benchmarker [razy][skrypt][skrypt]
↪[skrypt]...
```

Przykład

```
./script/performance/benchmarker 5 'Person.do_this' 'Person.
↪do_that'
```

profiler

Profiluje pojedynczą instrukcję w środowisku.

Użycie

```
./script/performance/profiler [skrypt][razy][flat|graph|
↪graph_html]
```

Przykład

```
./script/performance/profiler 'Person.do_this(10)' 25 graph
```

request

Skrypt opakowujący bibliotekę ruby-prof (*http://ruby-prof.rubyforge.org*). Skrypt ten pozwala wykonać większą liczbę żądań dla adresu URI w aplikacji i otrzymać szczegółowy raport profilu kodu w wersji tekstowej oraz HTML.

Użycie

```
./script/performance/request [opcje][ścieżka skryptu]
```

Opcje

`-n, --times [OOOO]`
: Określa, ile żądań należy przetworzyć (wartością domyślną jest `100`).

`-b, --benchmark`
: Test wydajności zamiast profilowania.

`--open [polecenie]`
: Polecenie otwarcia wyników profilowania (wartością domyślną jest „`open %s &`”).

Przykład

```
# Utworzenie pliku o nazwie 'perfscript' zawierającego
post('/sessions', { :login => 'berry', :password => 'test' })

# Wyniki
Thread ID: 218880
Total: 21.639647

%self total self wait child calls name
19.28 14.08 4.17 0.00 9.91 25200  Pathname#cleanpath
                                  _aggressive
17.11 5.36  3.70 0.00 1.66 157800 Pathname#chop
                                  _basename
5.38  2.22  1.16 0.00 1.05 50400  Pathname#
                                  initialize
4.66  1.50  1.01 0.00 0.49 265600 Kernel#===
```

```
 3.21  0.69  0.69 0.00 0.00 183000 Regexp#to_s
 3.20  0.72  0.69 0.00 0.03 377100 String#==
 3.18  1.01  0.69 0.00 0.32 800    Array#select
```

script/plugin

Menedżer dodatków Rails.

Użycie

```
./script/plugin [OPCJE] polecenie
```

Polecenia

`discover`
: Odnajduje repozytoria dodatków.

`list`
: Wyświetla dostępne dodatki.

`install`
: Instaluje dodatki ze znanych repozytoriów bądź adresów URL.

`update`
: Uaktualnia zainstalowane dodatki.

`remove`
: Odinstalowuje dodatki.

`source`
: Dodaje repozytorium kodu źródłowego dodatku.

`unsource`
: Usuwa repozytorium dodatku.

`sources`
: Wyświetla aktualnie skonfigurowane repozytoria dodatków.

script/process

Skrypty te służą do badania procesów oraz pomagają w ich kontrolowaniu.

inspector

Wyświetla informacje systemowe dotyczące dyspozytorów Rails (lub innych procesów wykorzystujących pliki *pid*) za pomocą polecenia `ps`.

Użycie

```
./script/process/inspector [opcje]
```

Opcje

`-s, --ps=`*polecenie*
: Domyślnie: `ps -o pid,state,user,start,time,pcpu,vsz,majplt,command -p %s`.

`-p, --pidpath=`*ścieżka*
: Domyślnie: */Users/berry/Sites/clearplay/tmp/pids*.

`-r, --pattern=`*wzorzec*
: Domyślnie: *dispatch.*.pid*.

Przykład

```
# własny ps, %s to gdzie pid się przeplata
inspector -s 'ps -o user,start,majflt,pcpu,vsz -p %s'
```

reaper

Skrypt `reaper` służy do ponownego uruchomienia, przeładowania, zwykłego zakończenia oraz zakończenia wymuszonego procesów wykonujących dyspozytor Rails (lub innych procesów odpowiadających na te same sygnały). Najczęściej robi się to, kiedy dostępna jest nowa wersja aplikacji, by można było uaktualnić istniejące procesy tak, by wykorzystywały one najnowszą wersję kodu.

Skrypt ten wykorzystuje pliki *pid* do pracy z procesami i domyślnie zakłada, że znajdują się one w katalogu *RAILS_ROOT/tmp/pids*.

Akcje skryptu `reaper` są następujące:

`restart`
: Ponownie uruchamia aplikację, przeładowując kod aplikacji oraz platformy.

`reload`
: Przeładowuje jedynie aplikację, ale nie platformę (na przykład środowisko programistyczne).

`graceful`
: Oznacza wszystkie procesy jako kandydatów do zakończenia po kolejnym żądaniu.

`kill`
: Wymusza zakończenie wszystkich procesów bez względu na to, czy akurat obsługują jakieś żądania.

Użycie

```
./script/process/reaper [opcje]
```

Opcje

`-a, --action=`*`nazwa`*
: `reload` | `graceful` | `kill` (domyślnie: `restart`).

`-p, --pidpath=`*`ścieżka`*
: Domyślnie: *[RAILS_ROOT]/tmp/pids*.

`-r, --pattern=`*`wzorzec`*
: Domyślnie: *dispatch.[0-9]*.pid*.

Przykład

```
# Wymuszone zakończenie wszystkich procesów przechowujących pliki pid w katalogu tmp/pids
reaper -a kill -r *.pid
```

spawner

Skrypt `spawner` opakowuje spawn-fcgi oraz Mongrel i ułatwia uruchamianie większej liczby procesów wykonujących dyspozytor Rails. Polecenie `spawn-fcgi` pochodzi z serwera lighttpd, jednak można je wykorzystać zarówno w lighttpd, jak i Apache (i dowolnym innym serwerze WWW obsługującym zarządzane zewnętrznie procesy FCGI). Mongrel jest zawarty automatycznie w `mongrel_rails`.

Użycie

```
./script/process/spawner [platforma][opcje]
```

Opcje

`-a, --address=ip`
: Wiązanie do adresu IP (domyślnie: `0.0.0.0`).

`-p, --port=numer`
: Numer portu początkowego (domyślnie: `8000`).

`-i, --instances=liczba`
: Liczba instancji (domyślnie: `3`).

`-r, --repeat=sekundy`
: Powtarza próbę wykonania skryptu co *n* sekund (domyślnie: wyłączone).

`-e, --environment=nazwa`
: `test | development | production` (domyślnie: `production`).

`-P, --prefix=ścieżka`
: Adres URL przedrostka dla aplikacji Rails (wykorzystywany jedynie w serwerze Mongrel w wersji 0.3.15 oraz wyższej).

`-n, --process=nazwa`
: Domyślnie: `dispatch`.

`-s, --spawner=`*`ścieżka`*
: Domyślnie: */usr/bin/env spawn-fcgi* (dla instalacji w systemie Mac OS X).

`-d, --dispatcher=`*`ścieżka`*
: Domyślnie: *[RAILS_ROOT]/public/dispatch.fcgi*.

Przykład

```
# Rozpoczyna 10 instancji, odliczając od 9100 do 9109, wykorzystując serwer Mongrel,
# jeśli jest on dostępny
./script/process/spawner -p 9100 -i 10
```

script/runner

Wykonuje kod w języku Ruby lub określony plik tego języka.

Użycie

```
./script/runner [opcje]('Jakiś.kod(Ruby)' lub nazwa_pliku)
```

Opcje

`-e, --environment=`*`nazwa`*
: Określa środowisko, w jakim ma działać skrypt `runner` (`test`, `development` bądź `production`).

Przykład

```
./script/runner MovieList.update_from_imdb -e production
```

> **Wskazówka**
>
> Skrypt `runner` można również wykorzystać do wykonywania kodu języka Ruby w skryptach powłoki:
>
> ```
> ------------------------------------------------------
> #!/usr/bin/env/Users/berry/Sites/myapp/script/runner
> Product.find(:all).each { |p| p.price *= 2 ; p.save! }
> ------------------------------------------------------
> ```

script/server

Uruchamia serwer WWW. Jeśli Mongrel jest zainstalowany, domyślnie zostanie uruchomiony. By wymusić użycie *webrick*, należy przekazać `webrick` jako opcję.

Użycie

```
./script/server [opcje]
```

Opcje

`-p, --port=`*port*
: Uruchamia Rails na określonym porcie (domyślnie: `3000`).

`-b, --binding=`*ip*
: Wiąże Rails z określonym adresem IP (domyślnie: `0.0.0.0`).

`-d, --daemon`
: Sprawia, że serwer działa jako daemon.

`-u, --debugger`
: Włącza debugowanie Ruby dla serwera.

`-e, --environment=`*nazwa*
: Określa środowisko, w jakim ma działać serwer (`test`, `development` bądź `production` — domyślnie `development`).

Środowiska

Środowiska Rails odzwierciedlają etapy tworzenia typowej aplikacji: programowanie, testowanie oraz produkcję.

Środowisko Rails można ustawić na jeden lub oba z poniższych sposobów:

- ustawienie zmiennej środowiskowej `RAILS_ENV` na nazwę środowiska (`development`, `test` lub `production`),