0
Рейтинг

Evrone

Подписаться

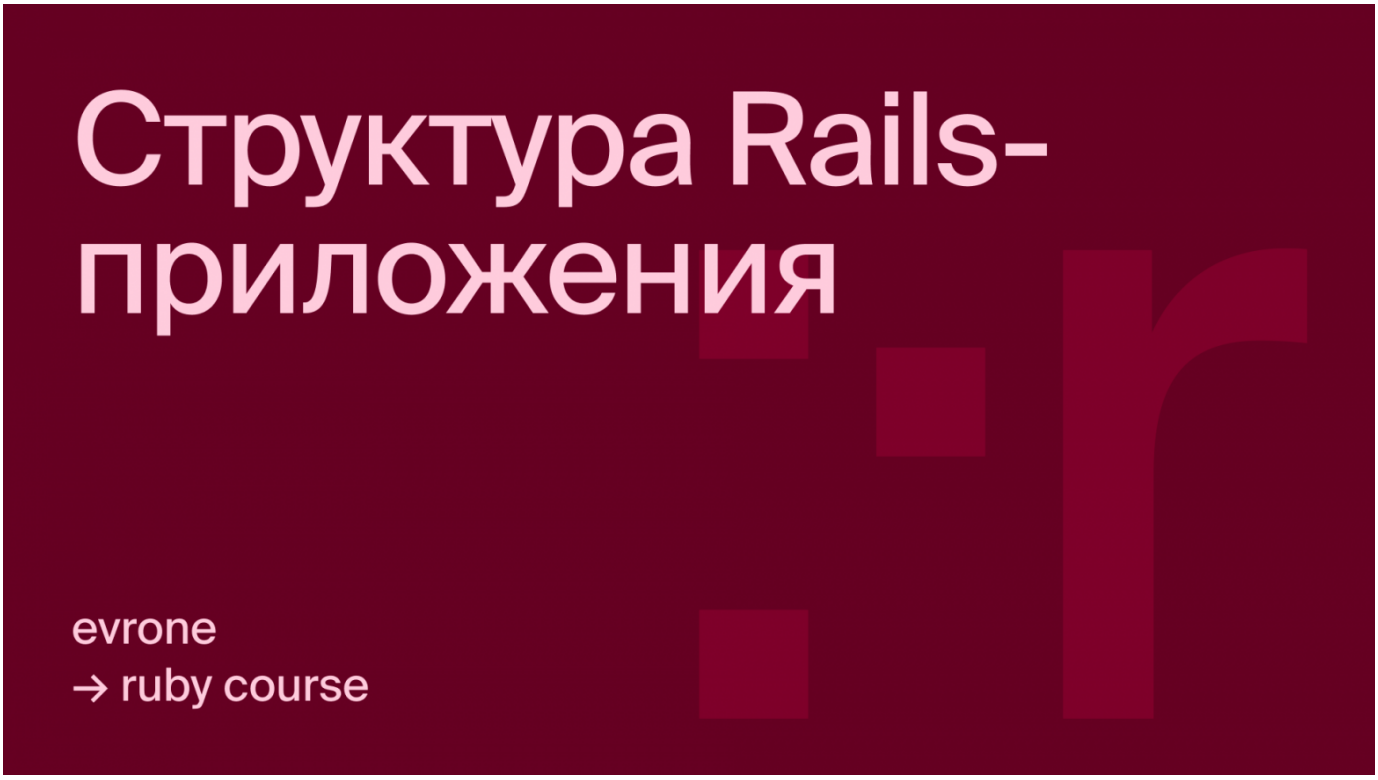
Evrone
21 окт 2022 в 19:33

Курс по Ruby+Rails. Часть 4. Структура Rails-приложения

8 мин 2.8K

Блог компании Evrone, Ruby*, Ruby on Rails*

Туториал



Сегодня начнём знакомство с Ruby on Rails. Для нашего курса это, пожалуй, самый важный инструмент, и следующие несколько лекций будут посвящены именно ему.

Сначала договоримся о терминах: мы не говорим «сайт». Вместо этого будем использовать термин «веб-приложение», потому что времена интернета с простыми статичными страничками прошли. Современные веб-приложения сложны. Для пользователя в них всегда

у разработчика немного. Однако под капотом прячется много движущихся частей: базы данных, генерация динамического контекста, внедрение и интеграция внешних элементов. Механическую работу программиста упрощают фреймворки. В случае Ruby — это Rails.

Ruby on Rails — это веб-фреймворк, который создали в компании «37signals» в рамках работы над внутренними инструментами. 37signals хотели разрабатывать веб-приложения быстрее и комфортнее — и создали Ruby on Rails.

Обычно программистские инструменты делают в виде библиотек или фреймворков. Инструменты этих разновидностей будут постоянно встречаться в вашей программистской жизни. Часто фреймворки и библиотеки путают, но разница есть. Библиотека — это код, к которому мы обращаемся из своего кода, и выполняем в нём. Фреймворк — это программная система, внутрь которой мы помещаем свой код, а она использует его при работе. Используя фреймворки мы получаем хорошо устроенное пространство, и гарантию, что всё будет работать как надо. А вот работу библиотеки в своём проекте, возможно, придется настраивать.

Фреймворк Ruby-on-rails нужен для удобной разработки сложных веб-приложений. Ключевое слово — сложных. Рельсы, как их ещё называют, значительно упрощают работу программиста.

Архитектурно Rails — это MVC-фреймворк. Запомним это название, саму конструкцию MVC мы разберём на одной из следующих лекций.

Как устроен Rails?

Чтобы установить rails на своей машине, используйте команду gem install rails. Чтобы создать будущее приложение, запустите генератор приложения. Настройки оставим по умолчанию — на 6-й рельсе они устанавливаются генератором достаточно разумно.

```
> gem install rails
> gem info rails

*** LOCAL GEMS ***
```

```
rails (6.1.4.1)
   Author: David Heinemeier Hansson
   Homepage: https://rubyonrails.org
   License: MIT
   Installed at (6.1.4.1): /Users/paul/.rbenv/versions/3.0.1/lib/ruby/gems/3.0.0

   Full-stack web application framework.
```

Генерацию нового приложения мы выполняем командой `rails new` с ключами `--skip-sprockets --skip-test --skip-system-test .` Эти ключи избавят нас от ненужных пока компонентов фреймворка. После ключей пишется имя будущего приложения. Подождём, пока Rails создаст приложение, установит все необходимые гемы и выполнит нужные настройки внутри приложения, — заметьте, что всё это делает одна единственная команда.

```
> rails new --skip-sprockets --skip-test --skip-system-test student_app

# ... очень много строк

> cd student_app
> tree -L 1 .
.
├── Gemfile
├── Gemfile.lock
├── README.md
├── Rakefile
├── app
├── babel.config.js
├── bin
├── config
├── config.ru
├── db
├── lib
├── log
├── node_modules
├── package.json
├── postcss.config.js
├── public
├── storage
├── tmp
├── vendor
└── yarn.lock
```

То, что запускаем – `rails new` – это т.н. генератор. Утилита командной строки, которая выполняет большую часть механической работы по установке необходимых для запуска приложения компонентов. Когда процесс инсталляции закончен, мы можем посмотреть, какую структуру приложения создал генератор в директории с помощью команды tree. Посмотрим в корень приложения: мы видим директории `app`, `bin`, `config`, `db`, `lib`, `log`, `node_modules`, `public`, `storage`, `tmp`, `vendor` и отдельные файлы, о которых поговорим чуть позже.

Директории – это контейнеры для основных компонентов приложения. Например, в директории `config` находится конфигурация приложения.

```
config
├── application.rb
├── boot.rb
├── cable.yml
├── credentials.yml.enc
├── database.yml
├── environment.rb
├── environments
├── initializers
├── locales
├── master.key
├── puma.rb
├── routes.rb
├── spring.rb
├── storage.yml
├── webpack
└── webpacker.yml
```

В директории `app` находится вся прикладная логика приложения: всё, что мы напрограммируем, будет там. Там будет своя структура поддиректорий.

```
app
├── assets
├── channels
├── controllers
├── helpers
├── javascript
├── jobs
├── mailers
```

```
├── models
└── views
```

В директории `bin` находятся утилиты и скрипты для запуска фреймворка и тестового приложения.

```
bin
├── bundle
├── rails
├── rake
├── setup
├── spring
├── webpack
├── webpack-dev-server
└── yarn
```

В директории `db` расположена информация об используемой базе данных.

```
db
└── seeds.rb
```

В директории `lib` находятся вспомогательные библиотеки, которые не относятся к прикладной логике приложения, но которые мы хотим переиспользовать в будущем. Например, в этой директории может находиться код, который нужным нам образом трансформирует объекты основных типов.

```
lib
├── assets
└── tasks
```

`Log` – директория для журналов, логов приложения.

```
log
└── development.log
```

В каталоге `node_modules` находятся зависимости, установленные `webpacker`'ом.

В директории `public` находятся файлы для статичного сервиса. Это html, css-файлы и собранные ассеты, которые отдаются веб-сервером без дополнительной обработки.

```
public
├── 404.html
├── 422.html
├── 500.html
├── apple-touch-icon-precomposed.png
├── apple-touch-icon.png
├── favicon.ico
└── robots.txt
```

`Storage` – это директория для хранения загружаемых файлов. Например, если ваше приложение позволяет загрузить в него картинки, то файлы по умолчанию попадут в этот каталог.

Директория `tmp` содержит временные файлы – кэши, предварительно собранные файлы для ассетов и скриптов и так далее.

```
tmp
├── cache
├── development_secret.txt
├── miniprofiler
├── pids
└── storage
```

`Vendor` – это специальная директория для установки библиотек сторонних поставщиков. например, можно настроить установку бандла приложения так, что гемы для него будут устанавливаться туда.

Директория app

Давайте посмотрим в директорию `app`, которая нам сейчас наиболее интересна – в основном код приложения хранится и разрабатывается в ней.

Рассмотрим самые важные для нас на текущем этапе поддиректории:

```
app
├── assets
│   ├── config
│   ├── images
│   └── stylesheets
├── channels
│   └── application_cable
├── controllers
│   ├── application_controller.rb
│   └── concerns
├── helpers
│   └── application_helper.rb
├── javascript
│   ├── channels
│   └── packs
├── jobs
│   └── application_job.rb
├── mailers
│   └── application_mailer.rb
├── models
│   ├── application_record.rb
│   └── concerns
└── views
    └── layouts
```

В `models` находятся модели – классы, которые описывают основные сущности данных, с которыми мы работаем в приложении.

Во `views` находятся шаблоны для будущих представлений веб-страниц, визуального контента, который будет представлять из себя наше приложение.

В `controllers` находятся контроллеры веб-приложения.

В `assets`, лежат исходники фронтенда приложения на js и css-файлы.

В `channels` находятся классы, которые работают с веб-сокетами, если они необходимы.

`Helpers` хранит хэлперы – вспомогательные модули, которые содержат методы, реализующие вспомогательное поведение для использования во `view`, отображениях.

В директории `javascript` программируется код сложных веб-клиентов. Например, мы разрабатываем SPA или работаем с реактивным фреймворком (Vue.js или React.js) поверх

нашего Ruby-приложения. Весь Ruby-код мы будем традиционно называть бэкендом. Бэкенд выполняется на сервере, в отличие от фронтенда — кода, который выполняется в браузере пользователя.

В `jobs` лежат задачи для отложенного выполнения.

`Mailer` — директория для мейлеров. Это специальные объекты, которые формируют и отправляют сообщения, пользуясь остальной логикой приложения. Например, сообщения для мессенджеров или электронной почты.

Такова базовая структура приложения. Мы пока не расшифровываем многие слова и термины, но будем объяснять, когда приступим непосредственно к их изучению.

Конфигурация проекта (в корневых файлах)

Поднимемся на директорию выше и поговорим о конфигурации. Это технически сложный раздел приложения. Сами видите — много директорий и новых слов, а когда мы это населим настоящими объектами с настоящим поведением — сложность ещё возрастёт.

Как же устроена конфигурация в Ruby-on-rails? Здесь реализована концепция «соглашение важнее настроек», то есть программист располагает свой код, а фреймворк находит его и включает в рантайм по заранее согласованным именам. Мы уже видели, что модели лежат в директории `models`. Значит модели и классы моделей фреймворк будет искать именно там.

Поэтому у нас нет избыточных файлов конфигурации. Всё уже достаточно умно организовано. Давайте посмотрим снова в корневую директорию на те 9 файлов, на которых мы не останавливались раньше:

```
.
├── Gemfile
├── Gemfile.lock
├── README.md
├── Rakefile
├── babel.config.js
├── config.ru
├── package.json
├── postcss.config.js
└── yarn.lock
```

`Gemfile` и `Gemfile.lock` — это файлы, в которых зафиксирован «бандл» — множество руби-библиотек или гемов, которые используются в нашем проекте. Там описаны

и изолированы зависимости. Это значит, что при добавлении новых библиотек мы должны внести их в `Gemfile`.

Файл `readme.md` – это классический файл документации, где находится описание проекта.

Также мы видим несколько файлов, которые настраивают WebPacker – JavaScript-фреймворк в нашем проекте. Это `babel.config.js`, `package.json`, `postcss.config.js`, а также `yarn.lock`.

Чуть больше мы увидим, если посмотрим напрямую в директорию `config`.

Здесь собрана описанная конфигурация фреймворка. Её относительно немного, файлы содержат минимально необходимые тонкие настройки.

```
config
├── application.rb
├── boot.rb
├── cable.yml
├── credentials.yml.enc
├── database.yml
├── environment.rb
├── environments
│   ├── development.rb
│   ├── production.rb
│   └── test.rb
├── initializers
│   ├── application_controller_renderer.rb
│   ├── backtrace_silencers.rb
│   ├── content_security_policy.rb
│   ├── cookies_serializer.rb
│   ├── filter_parameter_logging.rb
│   ├── inflections.rb
│   ├── mime_types.rb
│   ├── permissions_policy.rb
│   └── wrap_parameters.rb
├── locales
│   └── en.yml
├── master.key
├── puma.rb
├── routes.rb
├── spring.rb
├── storage.yml
├── webpack
```

```
│   ├── development.js
│   ├── environment.js
│   ├── production.js
│   └── test.js
└── webpacker.yml
```

`Application.rb` – это верхнеуровневая настройка приложения. Директория `environments` содержит настройки окружения, в которых выполняется фреймворк.

В базовом варианте окружений три. В процессе разработки мы работаем главным образом в `development` окружении. Запуск тестов происходит в `test`. Запуск готового приложения на сервере – это запуск приложения в окружении `production`.

В директории `initializers` находятся настройки реализации разных компонентов фреймворка. Не бойтесь сейчас что-то не запомнить, мы разберём все файлы подробнее, когда будем касаться их в следующих лекциях.

Ещё одна небольшая, но важная для понимания директория – `locales`. Здесь мы храним локализацию приложения, то есть настройки системы интернационализации – всё, что касается вывода фиксированных строк на разных языках. В файле `en.yml` находятся будущие английские строки для приложения. Русскоязычные строки мы поместим в файл `ru.yml`.

`Puma.rb` хранит конфигурацию сервера приложений puma, который запускается по умолчанию как базовый ruby-процесс. В его адресном пространстве происходит выполнение всего кода.

Ещё один очень важный файл – `routes.rb`. Там находятся маршруты, которые связывают HTTP-запросы с бизнес-логикой приложения.

В директории webpack мы видим настройки webpacker'а, в которые мы довольно редко залезаем руками. Если мы что-то и меняем, то в файле `webpacker.yml`, потому что он управляет сборкой всех javaScript-элементов.

Таково расположение основных элементов конфигурации приложения во фреймворке RoR.

Мы выполнили наш самый первый обзор rails-приложения. Узнали, что конфигурация расположения в нескольких файлах в корне проекта и директории `config`. Там хранятся тонкие настройки фреймворка. Отметим и запомним принцип «convention over configuration» – «соглашение важнее настроек», по которому выстраивается и конфигурация приложения, и весь его код в целом. Фреймворк RoR «знает», какие файлы

ему где искать, а программисту остаётся положить их в нужные места и правильно назвать.

Не стесняйтесь писать нам свои вопросы!

Теги: курсы программирования, ruby, ruby on rails, j ruby, обучение ruby

Хабы: Блог компании Evrone, Ruby, Ruby on Rails

Публикации

Искусство создания понятных графиков

Средний 7 мин 3.6K

Туториал Перевод

+38 79 2

dlinyj
20 часов назад

Измерение скорости чтения-записи носителей с помощью утилиты dd

Средний 11 мин 4K

Кейс

+38 48 29

SLY_G
10 часов назад

Известная, но очень странная кошачья повадка: кошки, приносящие игрушку — это эволюционная загадка

5 мин 7.8K

Перевод

+30 22 40

alizar
19 часов назад

Автономия разработчиков. Как устроены компании нового типа

Простой 7 мин 4K

Мнение

+25 20 21

Petr0v1
14 часов назад

Марс всё ближе: несмотря на проблемы, запуск Starship можно считать успешным

4 мин 4K

+16 10 12

ne_volkov
19 часов назад

Как в Ozon следят за чувствительной информацией в логах и при чем тут Толкиен?

Простой 10 мин 3.2K

Кейс

+15 22 3

Doctor_IT
15 часов назад

Сможет ли високосная минута решить проблему синхронизации часов?

6 мин 2.8K

+14 15 14

badcasedaily1
12 часов назад

Garbage Collection и JVM

Простой 17 мин 2.2K

Обзор

+13 68 3

SLY_G
16 часов назад

Дайджест научпоп-новостей за неделю, о которых мы ничего не писали

8 мин 1.6K

Дайджест

+13 5 1

Показать еще

ИНФОРМАЦИЯ

Сайт	evrone.ru
Дата регистрации	2 августа 2022
Дата основания	2008
Численность	101–200 человек
Местоположение	Россия

БЛОГ НА ХАБРЕ

19 мая в 19:04

Курс по Ruby+Rails. Часть 8. Модели и первые шаги

1.8K 0

25 апр в 14:29

Что нового в Proxmox 7.4

6.9K 23

6 апр в 14:00

Как добавить сторонние драйверы в установочный образ VMware ESXi 8

4.9K 18

22 мар в 19:40

Курс по Ruby+Rails. Часть 7. Модели и ActiveRecord

2.6K 1

27 фев в 19:55

Подробный гайд по Docker на M1

13K 6