0
Рейтинг

Evrone

Подписаться

Evrone
7 ноя 2022 в 21:39

Курс по Ruby+Rails. Часть 6. Роутинг и RESTful Applications

15 мин 2.2K

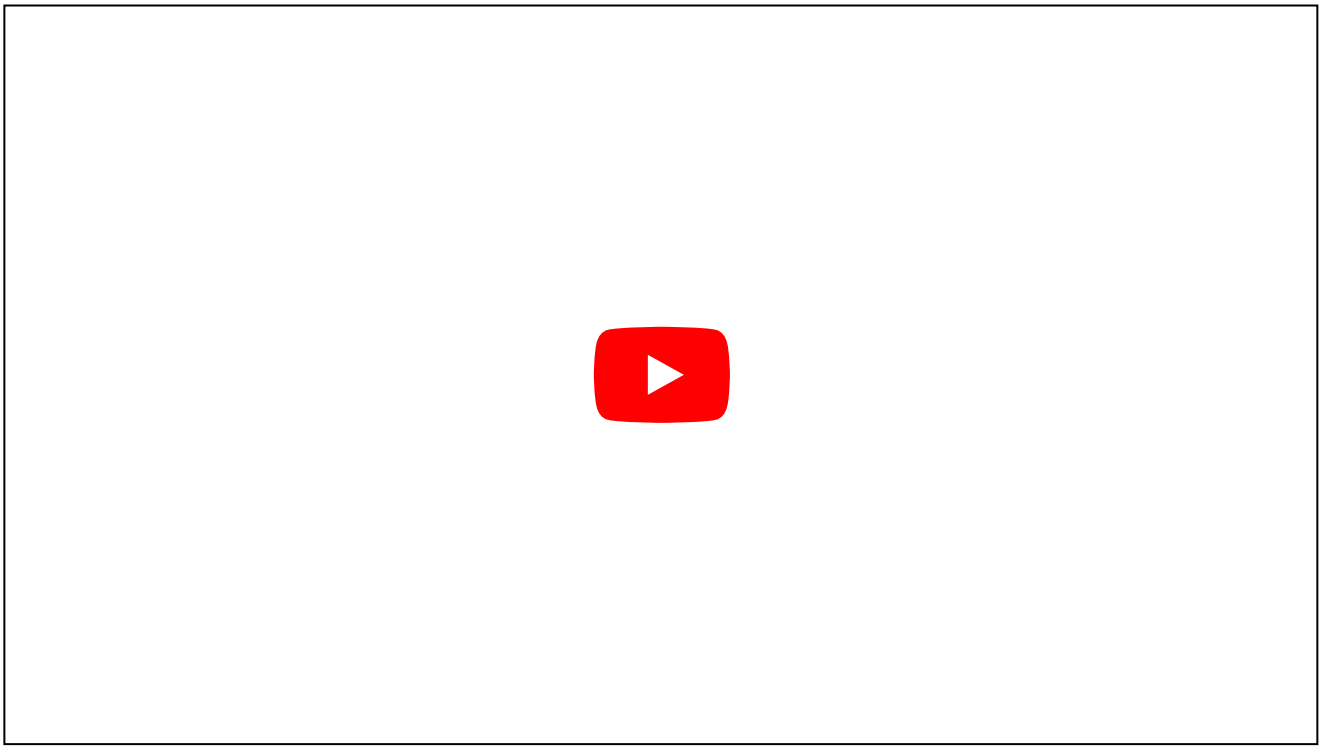
Блог компании Evrone, Ruby*, Ruby on Rails*

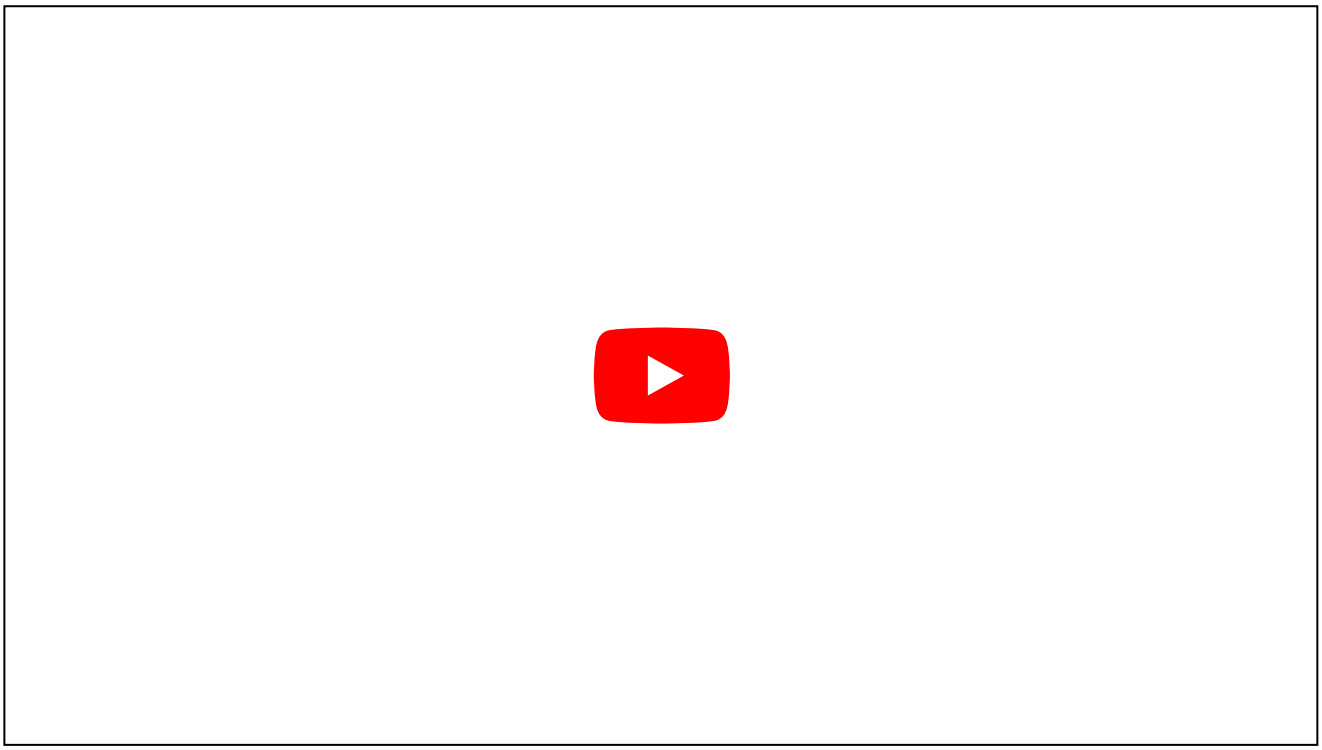
Туториал

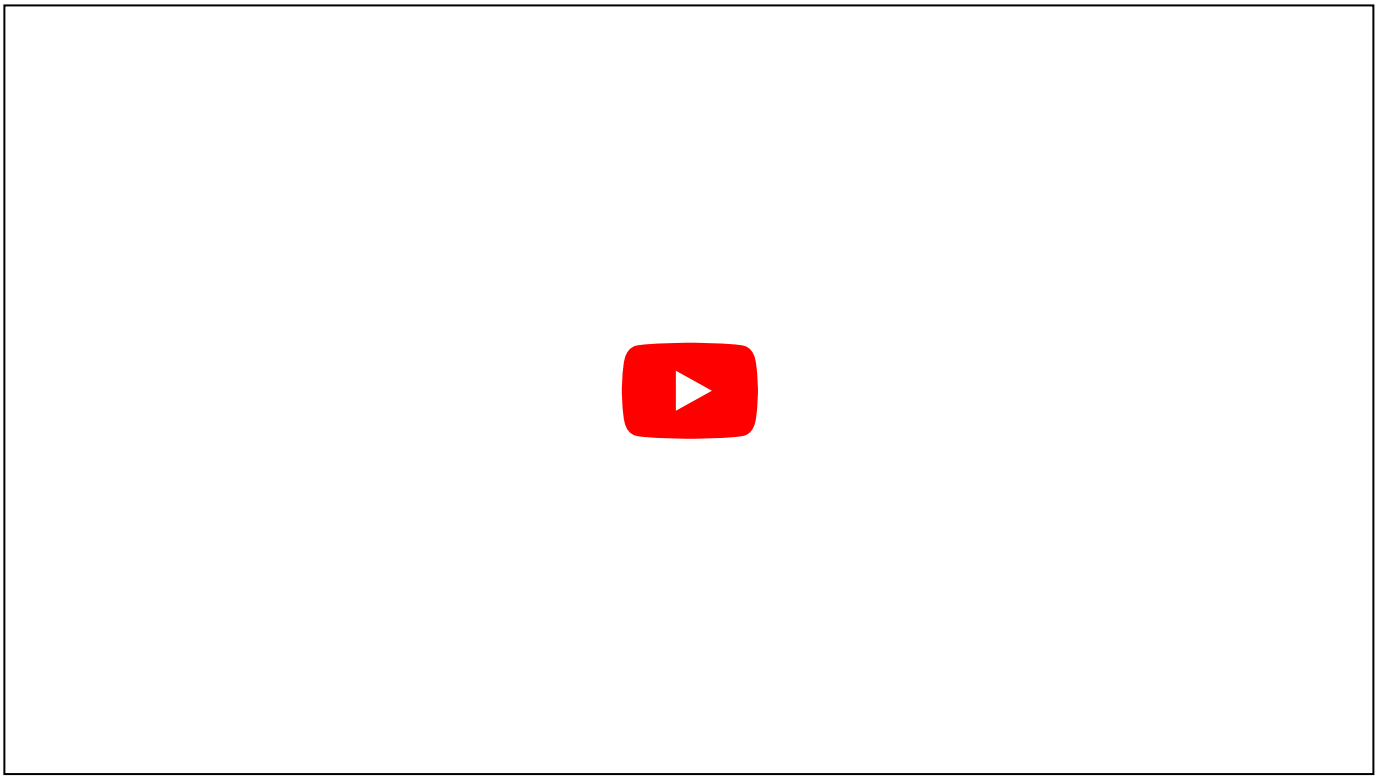
Сегодня мы поговорим о важнейшем элементе фреймворка Ruby on Rails – маршрутизации, а также посмотрим на принцип, вокруг которого она построена – архитектурный принцип REST.

Маршрутизация – это программное связывание элементов HTTP-запроса с конкретными элементами программного обеспечения сервера, которые выполняют этот запрос. Например,

в ответ на определенный глагол и путь запроса вызывается определенный метод (action) определенного контроллера, внутри которого производится обработка запроса.







Краткий обзор маршрутов

Чтобы понять это определение, необходимо вспомнить, из чего состоит стандартный HTTP-запрос. Вот его структура:

```
  PATCH    http://    enterprise.ru /employees/42
| глагол | протокол | хост         | путь (маршрут) -> ...
```

Здесь мы видим обозначение протокола, по которому происходит запрос, обозначение хоста, то есть имя сервера, которому передаётся запрос, и далее через слэш маршрут. Именно маршрут разбирается сервером приложения, чтобы доставить запрос в требуемый обработчик. Дополнительно для определения обработчика запроса, то есть для маршрутизации, используется так называемый глагол – в нашем случае PATCH. Глаголы иногда называют методами. Глаголы – это часть HTTP-запроса, определяющая смысл действия, которое будет произведено: чтение, запись, удаление данных и т.д.

Давайте рассмотрим частные примеры подобных запросов, а также, как фреймворк Ruby on Rails будет воспринимать их. Для краткости сейчас мы опустим имя хоста и частные параметры запроса – работать будем только с путями.

```
GET /employees
```

Этот запрос передаётся экшену `EmployeesController#index` для отображения списка сотрудников мероприятия – на структурном уровне это значит, что в классе `EmployeesController` Ruby on Rails находит для нас метод `index`.

Следующий пример:

```
POST /employees
```

Здесь мы видим глагол POST и тот же самый маршрут. Такой запрос Ruby on Rails передаст `EmployeesController`, но уже экшену `create`, чтобы создать запись о сотруднике.

Рассмотрим еще один пример:

```
PATCH /employees/42
```

Здесь маршрут тот же, однако после слэша указан целочисленный идентификатор. В таком виде Ruby on Rails воспримет запрос как параметр для экшена `edit` контроллера `EmployeesController`, чтобы редактировать запись о конкретном сотруднике.

Наконец, при помощи того же самого маршрута и глагола DELETE мы можем отправить серверу сообщение об удалении записи о конкретном сотруднике:

```
DELETE /employees/42
```

Как устроены маршруты?

Во-первых, вы видите описание некоторых ресурсов, в нашем случае – `employees`. То есть мы говорим о некоторых сущностях, называемых сотрудниками. Также мы говорим об идентификаторах конкретных сотрудников, а еще видим, что глагол формирует определенное представление, что предстоит сделать с конкретным ресурсом или совокупностью ресурсов.

Такой подход, иначе – архитектурный принцип расшифровки или интерпретации запроса, получил название REST. По принятому в разработке Ruby on Rails соглашению, а мы помним, что для нас соглашения важнее конфигурации, маршрутизация, то есть переправка запросов определенным экшенам в определенные контроллеры, происходит с соблюдение архитектурного стиля REST.

REST – это аббревиатура от **RE**presentational **S**tate **T**ransfer, архитектурный стиль, предназначенный для программирования взаимодействий клиента и сервиса по протоколу HTTP с передачей состояния по представлению. REST в общем случае ограничивает конструирование запросов и ответов определенными правилами. Эти правила делают систему взаимодействия клиент-сервис производительной, масштабируемой, простой, удобной для модификаций, переносимой, отслеживаемой и надежной.

Из чего «вырос» REST?

Теперь немного углубимся в архитектуру REST – подумаем, для чего она создавалась и что отражает.

В первую очередь, REST отвечает определенным требованиям, предпосылкам и обстоятельствам, в которых происходят HTTP-запросы. Например, в протоколе HTTP отсутствует состояние – то есть сервер не выполняет промежуточного хранения информации о статусе объектов между запросами. Разумеется, сервер хранит информацию в базе данных, но в ней хранится некоторая статичная информация о ресурсе между запросами, а вот состояния обработки и процесса трансформации объектов между запросами не сохраняются – это называется stateless-протокол.

Далее мы знаем, что в современном вебе необходимо кэшировать передаваемые данные. Запросы иногда достигают больших размеров, как и передаваемые данные (ими могут быть как текстовые файлы, так и графические, видеопотоки и множество иных разнообразных данных). Всё это необходимо предварительно буферизовать для быстрой выдачи клиентскому приложению. Кэширование – это сложная многоуровневая система, реализованная на базе разнородных сетей в интернете, и для нее важно иметь однородное представление о ресурсах.

Кроме всего прочего, мы знаем, что в мире множества существующих архитектурных решений сформировалась потребность на унификацию интерфейса HTTP-запросов. Например, в запросе желательно иметь однозначную идентификацию ресурса и действия с его участием. Важно иметь всю полноту данных для операций на конкретном ресурсе, а также – полноту метаданных, дополнительных данных о том, как необходимо осуществлять ответ на заданный запрос.

Представим, что нам необходимо сообщить удаленной стороне, что мы передаём или принимаем данные в формате HTML, XML, JSON. В REST мы работаем с понятием ресурса –

так что же такое ресурс? Это любой целостный объект для взаимодействия в веб-приложении. Иначе говоря, это совокупность данных о целом предмете взаимодействия.

В примерах вы видели слово employees – значит, мы взаимодействовали с ресурсом сотрудника – с множеством или с конкретным представителем, имеющим идентификатор 42. В архитектуре MVC ресурс определяется моделью данных, хранящейся в базе и трансформирующейся в приложении, а также множеством операций, выполняемых в контроллере. Далее к совокупности операций, объектов, взаимодействующих с ресурсом, относятся view, то есть отображение моделей в нужном пользователю виде, и любые специальные объекты, реализующие бизнес-логику приложения.

Архитектурный стиль REST позволяет сильно упростить задачу компоновки и поддерживаемости кода сложного веб-приложения. Во фреймворке Ruby on Rails именно REST используется как основное, базовое соглашение для разбивки кода на соответствующие ресурсам группы моделей-view-контроллеров.

REST характеризуется тем, что определяет смысл составных частей запроса. Давайте еще раз вернёмся к схеме запроса, чтобы посмотреть, как он трактуется соглашением REST:

```
 PATCH    http://    enterprise.ru /employees/42
| глагол | протокол | хост        | путь (маршрут) -> ...
```

Глагол запроса используется для того, чтобы однозначно определить метод взаимодействия с ресурсом. В данном случае нам необходимо изменить ресурс, поэтому мы используем глагол PATCH. Далее определяется протокол, по которому передаётся запрос. В нашем случае протокол HTTP. Затем появляется хост, то есть имя сервера, на котором выполнится запрос, и уже после – маршрут, то есть обозначение ресурса или ресурсов, с которыми мы взаимодействуем.

Разбираем механизмы маршрутизации в RoR на примерах

Теперь узнаем, с помощью каких механик эти маршруты, собранные в соответствии с REST-соглашением, переводятся в вызов конкретных методов и объектов.

Для разбора механик маршрутизации смоделируем студенческое приложение, иллюстрирующее работу в воображаемой интернет-академии. В директории `/config` находится файл `routes.rb`. Этот файл содержит написанные на специальном прикладном языке, или DSL, правила маршрутизации.

Правила маршрутизации – это описание соответствия запросов экшенам контроллеров. Все запросы в RoR проходят через контроллеры (за редким исключением т.н. запросов к

статическим файлам), а правила описывают маршруты в терминах ресурсов или, реже, соответствия конкретных путей url конкретным экшенам. Именно в файле `routes.rb` находится специфический язык, специфические объявления правил маршрутизации.

Мы будем вносить изменения в файл маршрутизации и проверять образовавшиеся маршруты командой `bin/rails routes`. Команда выполняется в корневой директории приложения. Сразу замечу: свежесгенерированное RoR-приложение может иметь до пары дюжин маршрутов по умолчанию – они образованы интегрированными компонентами фреймворка (системой пересылки сообщений, хранения файлов и.т.п.). Их мы для краткости будем игнорировать.

Примеры, которые мы сейчас разберём, сгенерированы при помощи команды `rails routes` с ключом “`-g`”, позволяющим сокращать вывод программы до определенных маршрутов, соответствующих описанным в виде регулярного выражения масок.

Итак, начинаем с чистого `routes.rb`. В нем определён лишь блок, внутри которого будут помещаться все маршрутные правила:

```
Rails.application.routes.draw do
end
```

Добавим корневой маршрут, запустим вывод маршрутов и посмотрим на результат:

```
Rails.application.routes.draw do
   root to: 'home#index'
end
```

```
> bin/rails routes
Prefix Verb URI Pattern Controller#Action
  root GET  /           home#index

# далее массив маршрутов по умолчанию
```

Как видите, `rails routes` вывел нам таблицу, в которой отражены поля т.н. префикса, далее – глагол веб-запроса, затем – URL, в этом поле располагается путь до необходимого места в приложении.

Заметим, что пути не включают в себя спецификацию протокола и названия хоста – только лишь относительный путь внутри самого приложения. В специальной нотации мы видим структуру « слово#слово ». Так в Ruby-документации обозначается вызов метода экземпляра объекта класса, прописанного перед `#` . В нашем случае перед `#` находится нормализованное название контроллера, а после `#` – action.

Получается, что в случае нашего примера в файле, определяющем класс `Home Controller` , находится метод `index` – экшен, запускаемый по достижении конкретного маршрута. Итак, мы сгенерировали корневой маршрут приложения, при обработке которого будет запущен экшен `index` контроллера `Home Controller` .

Теперь посмотрим, как будет выглядеть использование хелпера маршрута. В области видимости контроллеров и view образовался новый глобальный видимый метод – `root_path` . При его вызове будет генерироваться строка, содержащая URL-паттерн:

```
= link_to 'Academy', root_path
```

Созданный нами главный маршрут приложения – пример нересурсного маршрута: он отражает некоторое общее соглашение о формировании веб-страницы по корневому пути, но при этом не является указанием на специфический ресурс, т.е. совокупность конкретной модели и бизнес-логики вокруг нее. Обратите внимание, что в нересурсных маршрутах мы явно указываем имя контроллера.

Перейдем к ресурсным маршрутам, ведь именно они составляют основную «суперсилу» Ruby on Rails. Пользуясь правилом «convention over configuration» и архитектурными правилами REST, RoR реализует выразительный и лаконичный DSL маршрутизации ресурсов.

Создадим группу маршрутов для управления ресурсом студентов:

```
Rails.application.routes.draw do
   root to: 'home#index'

   resources :students
end
```

Теперь выведем команду `rails routes` :

```
      Prefix Verb   URI Pattern                       Controller#Action
        root GET    /                                 home#index

    students GET    /students(.:format)               students#index
             POST   /students(.:format)               students#create
 new_student GET    /students/new(.:format)           students#new
edit_student GET    /students/:id/edit(.:format)      students#edit
     student GET    /students/:id(.:format)           students#show
             PATCH  /students/:id(.:format)           students#update
             PUT    /students/:id(.:format)           students#update
             DELETE /students/:id(.:format)           students#destroy
```

Одна строка DSL-маршрута образовала целое семейство маршрутов. В данном случае мы видим полное множество маршрутов для объявленного ресурса. Ещё маршруты, служащие тому, чтобы видеть полный список студентов и создавать нового конкретного студента. Более того, здесь есть целый ряд маршрутов, управляющих записями о студентах, в том числе формы редактирования или удаления студента, а также пути загрузки полностью или частично измененной информации.

Вот подробная расшифровка действий сгенерированных хелперов:

```
GET    http://academy.edu/students      # students_path — показать список студентов
POST   http://academy.edu/students      # students_path — создать ресурс "студент"
GET    http://academy.edu/students/17   # student_path  — показать ресурс "студент"
PATCH  http://academy.edu/students/17   # student_path  — изменить ресурс "студент"
PUT    http://academy.edu/students/17   # student_path  — ЗАменить ресурс "студент"
DELETE http://academy.edu/students/17   # student_path  — удалить ресурс "студент"
```

Названия контроллеров и экшенов заранее предсказываются фреймворком, он ожидает, что в файлах лежат классы, именованные соответствующим образом.

Пример хелпера маршрута к странице редактирования студента выглядит так:

```
= link_to "Edit student ${@student.id}", edit_student_path(@student)
```

Давайте представим, что RoR-процесс отображает не HTML-страницы, а, например, JSON или JSON API. В таком случае нам не нужен весь массив возможных маршрутов — нужно

лишь ограниченное множество маршрутов. Мы можем уменьшить количество генерируемых RoR-маршрутов при помощи специального указания. Управлять списком ресурсов можно, добавляя к методу формирования маршрутов именованные аргументы `:only` и `:except`. Из соображений улучшения стиля, я рекомендую использовать именно `only`. Он создаёт четкий white list необходимых действий. Это важно при разработке больших приложений. Возьмём `only` и передадим ему список экшенов, которые необходимо сгенерировать. Соответственно, маршруты будут сгенерированы лишь для тех экшенов, которые мы опишем:

```
Rails.application.routes.draw do
  root to: 'home#index'

  resources :students, only: %i[show create edit]
end
```

Итак, мы запросили генерацию маршрутов ресурса `students` для только трех экшенов из всех возможных. Что же из этого вышло?

```
〉bin/rails routes -g student
      Prefix Verb URI Pattern                  Controller#Action
    students POST /students(.:format)          students#create
edit_student GET  /students/:id/edit(.:format) students#edit
     student GET  /students/:id(.:format)      students#show
```

Обратите внимание, что RoR автоматически генерирует маршруты с использованием определенных глаголов — согласно существующему соглашению. Например, для экшена `students create` образовался глагол POST, это значит, что только такой глагол будет обрабатываться на этом маршруте. Если мы употребим GET вместо него — получим ошибку 404.

Иногда необходимо объявлять маршруты для действий над всей коллекцией ресурсов. Экшены, к которым будут создаваться маршруты, не входят в множество стандартных. Вернемся к студентам — допустим, нам необходимо отправлять студентов на удаленку или назначать каждому из них курс по Ruby. Выглядеть это будет так:

```
Rails.application.routes.draw do
  root to: 'home#index'

  resources :students, only: %i[show create edit] do
    post :assign_to_ruby_course, on: :member
```

```
    post :send_to_remote, on: :collection
  end
end
```

Мы открываем блок в выражении объявления ресурсов и добавляем необходимые действия. Это делается при помощи глагола, обозначающего действие, префикса и пометки элемента коллекции ресурсов, на котором необходимо выполнить действие. Как видите, курс по Ruby мы назначаем индивидуально, а на удаленку отправляем всю коллекцию.

С этим уточнением у нас образовался новый маршрут:

```
                      Prefix Verb URI Pattern                                       Co
assign_to_ruby_course_student POST /students/:id/assign_to_ruby_course(.:format) st
                    students POST /students(.:format)                             st
                edit_student GET  /students/:id/edit(.:format)                    st
                     student GET  /students/:id(.:format)                         st
```

Если нам нужно ввести в приложение несколько ресурсов, над которыми возможны одинаковые действия, на помощь приходят `concerns` – обобщения маршрутов. Например, у нас в академии появились преподаватели, к которым применимы все действия, уже описанные для студентов – т.е. нужны те же специальные маршруты. Во избежание избыточного кода мы можем формировать `concerns`. Преобразуем код в `concerns`:

```
Rails.application.routes.draw do
  root to: 'home#index'

  resources :students, only: %i[show create edit] do
    post :assign_to_ruby_course, on: :member
    post :send_to_remote, on: :collection
  end

  resources :trainers, only: %i[show create edit] do
    post :assign_to_ruby_course, on: :member
    post :send_to_remote, on: :collection
  end
end
```

Эту длинную запись при помощи консёрна можно сократить вот так:

```
Rails.application.routes.draw do
  root to: 'home#index'

  concern :assignable do
    post :assign_to_ruby_course, on: :member
    post :send_to_remote, on: :collection
  end

  resources :students, only: %i[show create edit],
                       concerns: :assignable

  resources :trainers, only: %i[show create edit],
                       concerns: :assignable
end
```

Мы выделили в консёрн `assignable` – т.е. повторяющиеся маршруты, а затем объявили использование обобщения для множеств ресурсов студентов и преподавателей. Вывод выглядит следующим образом:

```
〉bin/rails routes -g "(student)|(trainer)"
                      Prefix Verb URI Pattern                                       Co
assign_to_ruby_course_student POST /students/:id/assign_to_ruby_course(.:format) st
      send_to_remote_students POST /students/send_to_remote(.:format)             st
                     students POST /students(.:format)                            st
                 edit_student GET  /students/:id/edit(.:format)                   st
                      student GET  /students/:id(.:format)                        st
assign_to_ruby_course_trainer POST /trainers/:id/assign_to_ruby_course(.:format) tr
      send_to_remote_trainers POST /trainers/send_to_remote(.:format)             tr
                     trainers POST /trainers(.:format)                            tr
                 edit_trainer GET  /trainers/:id/edit(.:format)                   tr
                      trainer GET  /trainers/:id(.:format)                        tr
```

Время от времени, а на деле – довольно часто, у нас возникает потребность обращаться к ресурсам опосредованно, т.е. через другие ресурсы. В таком случае ресурсы, находящиеся внутри других ресурсов, называются вложенными.

Например, у каждого из наших студентов может быть собственный массив оценок, для которых необходимо реализовать RESTfull API. Здесь мы уже не будем приводить весь файл – он становится достаточно толстым. В этом и последующих примерах будем пользоваться только фрагментами:

```
resources :students, only: %i[show create edit] do
  concerns :assignable
  resources :marks, only: %i[create destroy]
end
```

Итак, мы чуть-чуть переписали использование обобщения, и заодно внутри множества студентов объявили множество ресурсов оценки – ограничили действия с ними до постановки и удаления. Сгенерировались следующие маршруты:

```
       Prefix Verb   URI Pattern                                  Controller#Action
student_marks POST   /students/:student_id/marks(.:format)      marks#create
 student_mark DELETE /students/:student_id/marks/:id(.:format) marks#destroy
```

Мы видим, что внутри маршрута студентов образовался маршрут к оценке конкретного экземпляра.

В образованных нами маршрутах есть слова, начинающиеся с двоеточия – это подстановочные параметры. Таким образом RoR сообщает, что в образованном экшене будет передан параметр с конкретным именем, куда RoR подставит идентификатор конкретного студента, внутри маршрута которого мы вложили ресурс оценки.

Обратите внимание, что во втором случае у нас передаются два подстановочных параметра. Первый – `student id`, идентификатор студента, второй – `mark id`, идентификатор оценки.

Двинемся дальше. Если нам необходимо по логике приложения работать с вложенным ресурсом со ссылкой к родительскому ресурсу, тогда RoR использует полную форму обращения, но если нужно работать только с вложенным ресурсом – мы воспользуемся сокращением. Чтобы сократить строку вложенного маршрута до необходимого приложению минимума, нужно использовать аргумент `shallow`:

```
resources :students, only: %i[show create edit] do
  concerns :assignable
  resources :marks, only: %i[create destroy], shallow: true
end
```

Мы пометили ресурс оценки как мелкий ресурс и теперь посмотрим, как выглядят маршруты:

```
       Prefix Verb   URI Pattern                              Controller#Action
student_marks POST   /students/:student_id/marks(.:format) marks#create
         mark DELETE /marks/:id(.:format)                    marks#destroy
```

Благодаря такому решению, маршрут оценки сократился – в нем отсутствует длинный префикс, свидетельствующий непосредственно о вложении. Полная и сокращенная маршрутизация используются по усмотрению – в зависимости от требования заказчика или логики приложения.

В RoR предусмотрено выделение маршрутов в пространство имен. Например, нам нужно создать админку с отдельными маршрутами и контроллерами для них. Бизнес-логика админки отличается от логики простого приложения, и для админки потребуются как специальные маршруты, так и специальные контроллеры. Функционально они должны быть похожи с уже созданными, но по факту отличаться, чтобы не путать логику двух разных частей приложения. Физически это будет реализовано через модули и вложенные в них классы контроллеров. Опишем мы это так:

```
namespace :admin do
  resources :students do
    resources :marks
  end
  resources :trainers
end
```

Мы объявили `namespace` и поместили в него ресурсы с необходимой вложенностью. Маршруты и область имен у нас получаются следующие:

```
                Prefix Verb   URI Pattern
    admin_student_marks GET    /admin/students/:student_id/marks(.:format)
                        POST   /admin/students/:student_id/marks(.:format)
 new_admin_student_mark GET    /admin/students/:student_id/marks/new(.:format)
edit_admin_student_mark GET    /admin/students/:student_id/marks/:id/edit(.:format)
```

Префиксы наших хелпер-методов пропорционально усложняются – они находятся слева в таблице. У нас появился общий для всех сгенерированных маршрутов префикс `admin`. Также мы видим в колонке `ControllerAction` нотацию со слэшем, означающую, что

`marks controller` будет находиться внутри `namespace admin`. Таким образом, мы создали отдельный контроллер общей логики.

Если нужно выделить специальный префикс для маршрута, контроллер которого нет смысла помещать в собственное пространство имен, можно использовать метод `scope` в следующем виде:

```
scope :statistics do
  resources :marks, only: %i[] do
    get :statistics, on: :collection
  end
end
```

Например, мы сделали префикс для маршрутов отображения статистики, поместили в него хорошо знакомый ресурс оценок, соответствующий контроллер, определили для него что-то ограниченное, запретив формировать маршруты по умолчанию. Давайте посмотрим, что в таком случае образуется в маршруте:

```
          Prefix Verb URI Pattern                                Controller#Action
statistics_marks GET  /statistics/marks/statistics(.:format) marks#statistics
```

В URL появляется префикс маршрута `statistics`, при этом `marks controller` находится в общем пространстве, корневом или иначе – `top level`.

Если контроллер наоборот нужно убрать в выделенное пространство имен, но не формировать префикс, используется другая форма выражения – `scope module`:

```
scope module: :accounting do
  resources :trainers, only: %i[] do
    get :salary
  end
end
```

Мы видим, что `scope` объявлен для модуля с названием `accounting`. Внутри него находится контроллер для преподавателей – например, с целью управления их зарплатой. Подобное описание формирует следующие маршруты:

```
        Prefix Verb URI Pattern                                Controller#Action
trainer_salary GET  /trainers/:trainer_id/salary(.:format) accounting/trainers#sala
```

В правой колонке для контроллера образуется пространство имен, в которое нужно поместить `trainers controller` – этот модуль отличается от предыдущих, для него нужны специальные контроллеры, и бизнес-логика в нем будет соответствующая. В то же самое время, в паттернах маршрута приложения мы не видим специального обособления – лишь то, что в маршрутах для преподавателей появились уточняющие моменты.

Напоследок, RoR позволяет определять маршруты для ресурсов в единственном числе. Добавим новый маршрут к информации о ректоре:

```
resource :rector, only: :show
```

Учтите, что теперь все слова мы используем в единственном числе. В итоге образуется новый маршрут:

```
Prefix Verb URI Pattern       Controller#Action
rector GET  /rector(.:format) rectors#show
```

Учтите, что даже в этом случае соглашение RoR диктует необходимость именования контроллера во множественном числе ресурса – несмотря на то, что в логике приложения и в маршрутах ректор будет присутствовать в единственном числе.

Не стесняйтесь задавать вопросы в комментариях, мы обязательно ответим.

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

Хабы: Блог компании Evrone, Ruby, Ruby on Rails

Сайт Сайт Сайт Сайт

24 Карма 0 Рейтинг

Evrone @Evrone

Пользователь

Подписаться 

Комментарии 2

Публикации

ЛУЧШИЕ ЗА СУТКИ ПОХОЖИЕ

mr-pickles
18 часов назад

16-, 8- и 4-битные форматы чисел с плавающей запятой

Средний 15 мин 6.7K

+68 88 28

ru_vds
15 часов назад

Искусство создания понятных графиков

Средний 7 мин 3.5K

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 7.6K

Перевод

+30 22 38

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 11

Doctor_IT
14 часов назад

Сможет ли високосная минута решить проблему синхронизации часов?

6 мин 2.7K

+15 15 14

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

badcasedaily1
12 часов назад

Garbage Collection и JVM

Простой 17 мин 2.1K

Обзор

+13 67 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

Техническая поддержка