**Цель роутера Rails**

Роутер Rails распознает URL и соединяет их с экшном контроллера. Он также создает пути и URL, избегая необходимость писать тяжелый код в ваших вьюхах.

**Соединение URL с кодом**

Когда ваше приложение на Rails получает входящий запрос для:

|  |
| --- |
| GET /patients/17 |

оно опрашивает роутер на предмет соответствия экшну контроллера. Если первый соответствующий маршрут это:

|  |
| --- |
| get '/patients/:id', to: 'patients#show' |

то запрос будет направлен в контроллер patients в экшн show с { id: '17' } в params.

**Создание URL из кода**

Также можно создавать пути и URL. Если вышеуказанный маршрут изменить как:

|  |
| --- |
| get '/patients/:id', to: 'patients#show', as: 'patient' |

и ваше приложение содержит код в контроллере:

|  |
| --- |
| @patient = Patient.find(17) |

и это в соответствующей вьюхе:

<%= link\_to 'Patient Record', patient\_path(@patient) %>

тогда роутер создаст путь /patients/17. Это увеличит устойчивость вашей вьюхи и упростит код для понимания. Отметьте, что id не нужно указывать в маршрутном хелпере.

**Ресурсный роутинг**

Ресурсный роутинг позволяет быстро объявлять все обычные маршруты для заданного ресурсного контроллера. Вместо объявления отдельных маршрутов для экшнов index, show, new, edit, create, update и destroy, ресурсный маршрут объявляет их в единственной строке кода.

**Ресурсы в вебе**

Браузеры запрашивают страницы от Rails, выполняя запрос по URL, используя определенный метод HTTP, такой как GET, POST, PATCH, PUT и DELETE. Каждый метод - это запрос на выполнение операции с ресурсом. Ресурсный маршрут соединяет несколько родственных запросов с экшнами в одном контроллере.

Когда приложение на Rails получает входящий запрос для:

DELETE /photos/17

оно просит роутер соединить его с экшном контроллера. Если первый соответствующий маршрут такой:

resources :photos

Rails переведет этот запрос в метод destroy контроллера photos с { id: '17' } в params.

**CRUD, методы и экшны**

В Rails ресурсный маршрут предоставляет соединение между методами HTTP и URL к экшнам контроллера. По соглашению, каждый экшн также соединяется с определенной операцией CRUD в базе данных. Одна запись в файле роутинга, такая как:

resources :photos

создает семь различных маршрутов в вашем приложении, все соединенные с контроллером Photos:

|  |  |  |  |
| --- | --- | --- | --- |
| **Метод HTTP** | **Путь** | **Экшн** | **Использование** |
| GET | /photos | index | отображает список всех фото |
| GET | /photos/new | new | возвращает форму HTML для создания нового фото |
| POST | /photos | create | создает новое фото |
| GET | /photos/:id | show | отображает определенное фото |
| GET | /photos/:id/edit | edit | возвращает форму HTML для редактирования фото |
| PATCH/PUT | /photos/:id | update | обновляет определенное фото |
| DELETE | /photos/:id | destroy | удаляет определенное фото |

Поскольку роутер использует как метод HTTP, так и URL, для сопоставления с входящими запросами, четыре URL соединяют с семью различными экшнами.

Маршруты Rails сравниваются в том порядке, в котором они определены, поэтому, если имеется resources :photos до get 'photos/poll' маршрут для экшна show в строке resources совпадет до строки get. Чтобы это исправить, переместите строку get **над** строкой resources, чтобы она сравнивалась первой.

**Путь и хелперы URL**

Создание ресурсного маршрута также сделает доступными множество хелперов в контроллере вашего приложения. В случае с resources :photos:

* photos\_path возвращает /photos
* new\_photo\_path возвращает /photos/new
* edit\_photo\_path(:id) возвращает /photos/:id/edit (например, edit\_photo\_path(10) возвращает /photos/10/edit)
* photo\_path(:id) возвращает /photos/:id (например, photo\_path(10) возвращает /photos/10)

Каждый из этих хелперов имеет соответствующий хелпер \_url (такой как photos\_url), который возвращает тот же путь с добавленными текущими хостом, портом и префиксом пути.

**Определение нескольких ресурсов одновременно**

Если необходимо создать маршруты для более чем одного ресурса, можете сократить ввод, определив их в одном вызове resources:

resources :photos, :books, :videos

Это приведет к такому же результату, как и:

resources :photos

resources :books

resources :videos

**Одиночные ресурсы**

Иногда имеется ресурс, который клиенты всегда просматривают без ссылки на ID. Обычный пример, /profile всегда показывает профиль текущего вошедшего пользователя. Для этого можно использовать одиночный ресурс, чтобы связать /profile (а не /profile/:id) с экшном show:

get 'profile', to: 'users#show'

Передавая String в match ожидается следующий формат - controller#action, при передаче Symbol произойдёт непосредственно прямое обращение к экшену:

get 'profile', to: :show

Этот ресурсный маршрут:

resource :geocoder

создаст шесть различных маршрутов в вашем приложении, все связанные с контроллером Geocoders:

|  |  |  |  |
| --- | --- | --- | --- |
| **Метод HTTP** | **Путь** | **Экшн** | **Использование** |
| GET | /geocoder/new | new | возвращает форму HTML для создания нового геокодера |
| POST | /geocoder | create | создает новый геокодер |
| GET | /geocoder | show | отображает один и только один ресурс геокодера |
| GET | /geocoder/edit | edit | возвращает форму HTML для редактирования геокодера |
| PATCH/PUT | /geocoder | update | обновляет один и только один ресурс геокодера |
| DELETE | /geocoder | destroy | удаляет ресурс геокодера |

Поскольку вы можете захотеть использовать один и тот же контроллер и для одиночного маршрута (/account), и для множественного маршрута (/accounts/45), одиночные ресурсы ведут на множественные контроллеры. По этой причине, например, resource :photo и resources :photos создадут и одиночные, и множественные маршруты, привязанные к одному и тому же контроллеру (PhotosController).

Одиночный ресурсный маршрут создает эти хелперы:

* new\_geocoder\_path возвращает /geocoder/new
* edit\_geocoder\_path возвращает /geocoder/edit
* geocoder\_path возвращает /geocoder

Как и в случае с множественными ресурсами, те же хелперы, оканчивающиеся на \_url также включают хост, порт и префикс пути.

Давняя ошибка мешает form\_for работать автоматически с одиночными ресурсными маршрутами. Для решения данной проблемы, указывайте URL для формы, вот так:

form\_for @geocoder, url: geocoder\_path do |f|

**Пространство имен контроллера и роутинг**

Возможно, вы захотите организовать группы контроллеров в пространство имен. Чаще всего группируют административные контроллеры в пространство имен Admin::. Следует поместить эти контроллеры в директорию app/controllers/admin и затем можно сгруппировать их вместе в роутере:

namespace :admin do

resources :posts, :comments

end

Это создаст ряд маршрутов для каждого контроллера posts и comments. Для Admin::PostsController, Rails создаст:

|  |  |  |  |
| --- | --- | --- | --- |
| **Метод HTTP** | **Путь** | **Экшн** | **Именнованнный хелпер** |
| GET | /admin/posts | index | admin\_posts\_path |
| GET | /admin/posts/new | new | new\_admin\_post\_path |
| POST | /admin/posts | create | admin\_posts\_path |
| GET | /admin/posts/:id | show | admin\_post\_path(:id) |
| GET | /admin/posts/:id/edit | edit | edit\_admin\_post\_path(:id) |
| PATCH/PUT | /admin/posts/:id | update | admin\_post\_path(:id) |
| DELETE | /admin/posts/:id | destroy | admin\_post\_path(:id) |

Если хотите маршрут /photos (без префикса /admin) к Admin::PostsController, можете использовать:

scope module: 'admin' do

resources :posts, :comments

end

или для отдельного случая:

resources :posts, module: 'admin'

Если хотите маршрут /admin/photos к PostsController (без префикса модуля Admin::), можно использовать:

scope '/admin' do

resources :posts, :comments

end

или для отдельного случая:

resources :posts, path: '/admin/posts'

В каждом из этих случаев, именнованные маршруты остаются теми же, что и без использования scope. В последнем случае, следующие пути соединят с PostsController:

|  |  |  |  |
| --- | --- | --- | --- |
| **Метод HTTP** | **Путь** | **Экшн** | **Именнованнный хелпер** |
| GET | /admin/posts | index | posts\_path |
| GET | /admin/posts/new | new | new\_post\_path |
| POST | /admin/posts | create | posts\_path |
| GET | /admin/posts/:id | show | post\_path(:id) |
| GET | /admin/posts/:id/edit | edit | edit\_post\_path(:id) |
| PUT | /admin/posts/:id | update | post\_path(:id) |
| DELETE | /admin/posts/:id | destroy | post\_path(:id) |

**Вложенные ресурсы**

Нормально иметь ресурсы, которые логически подчинены другим ресурсам. Например, предположим ваше приложение включает эти модели:

class Magazine < ActiveRecord::Base

has\_many :ads

end

class Ad < ActiveRecord::Base

belongs\_to :magazine

end

Вложенные маршруты позволяют перехватить эти отношения в вашем роутинге. В этом случае можете включить такое объявление маршрута:

resources :magazines do

resources :ads

end

В дополнение к маршрутам для magazines, это объявление также создаст маршруты для ads в AdsController. URL с ad требует magazine:

|  |  |  |  |
| --- | --- | --- | --- |
| **Метод HTTP** | **Путь** | **Экшн** | **Использование** |
| GET | /magazines/:magazine\_id/ads | index | отображает список всей рекламы для определенного журнала |
| GET | /magazines/:magazine\_id/ads/new | new | возвращает форму HTML для создания новой рекламы, принадлежащей определенному журналу |
| POST | /magazines/:magazine\_id/ads | create | создает новую рекламу, принадлежащую указанному журналу |
| GET | /magazines/:magazine\_id/ads/:id | show | отражает определенную рекламу, принадлежащую определенному журналу |
| GET | /magazines/:magazine\_id/ads/:id/edit | edit | возвращает форму HTML для редактирования рекламы, принадлежащей определенному журналу |
| PATCH/PUT | /magazines/:magazine\_id/ads/:id | update | обновляет определенную рекламу, принадлежащую определенному журналу |
| DELETE | /magazines/:magazine\_id/ads/:id | destroy | удаляет определенную рекламу, принадлежащую определенному журналу |

Также будут созданы маршрутные хелперы, такие как magazine\_ads\_url и edit\_magazine\_ad\_path. Эти хелперы принимают экземпляр Magazine как первый параметр (magazine\_ads\_url(@magazine)).

**Ограничения для вложения**

Вы можете вкладывать ресурсы в другие вложенные ресурсы, если хотите. Например:

resources :publishers do

resources :magazines do

resources :photos

end

end

Глубоко вложенные ресурсы быстро становятся громоздкими. В этом случае, например, приложение будет распознавать URL, такие как:

/publishers/1/magazines/2/photos/3

Соответствующий маршрутный хелпер будет publisher\_magazine\_photo\_url, требующий определения объектов на всех трех уровнях. Действительно, эта ситуация достаточно запутана, так что в статье Jamis Buck предлагает правило хорошей разработки на Rails:

*Ресурсы никогда не должны быть вложены глубже, чем на 1 уровень.*

**Мелкое вложение**

Один из способов избежать глубокого вложения (как показано выше) является создание экшнов коллекции в пространстве имен родителя, чтобы чувствовать иерархию, но не вкладывать экшны элементов. Другими словами, создавать маршруты с минимальным количеством информации для однозначной идентификации ресурса, например так:

resources :posts do

resources :comments, only: [:index, :new, :create]

end

resources :comments, only: [:show, :edit, :update, :destroy]

Эта идея балансирует на грани между наглядностью маршрутов и глубоким вложением. Существует сокращенный синтаксис для получения подобного, с помощью опции :shallow:

resources :posts do

resources :comments, shallow: true

end

Это создаст те же самые маршруты из первого примера, Также можно определить опцию :shallow в родительском ресурсе, в этом случае все вложенные ресурсы будут мелкие:

resources :posts, shallow: true do

resources :comments

resources :quotes

resources :drafts

end

Метод shallow в DSL создает пространство имен, в котором каждое вложение мелкое. Это создаст те же самые маршруты из предыдущего примера:

shallow do

resources :posts do

resources :comments

resources :quotes

resources :drafts

end

end

Также существуют две опции для scope для настройки мелких маршрутов. :shallow\_path добавляет префикс к путям элемента из указанного параметра:

scope shallow\_path: "sekret" do

resources :posts do

resources :comments, shallow: true

end

end

Для ресурса комментариев будут созданы следующие маршруты:

|  |  |  |
| --- | --- | --- |
| **Метод HTTP** | **Путь** | **Именнованный хелпер** |
| GET | /posts/:post\_id/comments(.:format) | post\_comments |
| POST | /posts/:post\_id/comments(.:format) | post\_comments |
| GET | /posts/:post\_id/comments/new(.:format) | new\_post\_comment |
| GET | /sekret/comments/:id/edit(.:format) | edit\_comment |
| GET | /sekret/comments/:id(.:format) | comment |
| PATCH/PUT | /sekret/comments/:id(.:format) | comment |
| DELETE | /sekret/comments/:id(.:format) | comment |

Опция :shallow\_prefix добавляет указанный параметр к именнованным хелперам:

scope shallow\_prefix: "sekret" do

resources :posts do

resources :comments, shallow: true

end

end

Для ресурса комментариев будут созданы следующие маршруты:

|  |  |  |
| --- | --- | --- |
| **Метод HTTP** | **Путь** | **Именнованный хелпер** |
| GET | /posts/:post\_id/comments(.:format) | post\_comments |
| POST | /posts/:post\_id/comments(.:format) | post\_comments |
| GET | /posts/:post\_id/comments/new(.:format) | new\_post\_comment |
| GET | /comments/:id/edit(.:format) | edit\_sekret\_comment |
| GET | /comments/:id(.:format) | sekret\_comment |
| PATCH/PUT | /comments/:id(.:format) | sekret\_comment |
| DELETE | /comments/:id(.:format) | sekret\_comment |

**Концерны маршрутов**

Концерны маршрутов (Routing Concerns) позволяют объявлять обычные маршруты, которые затем могут быть повторно использованы внутри других ресурсов и маршрутов. Чтобы определить концерн:

concern :commentable do

resources :comments

end

concern :image\_attachable do

resources :images, only: :index

end

Эти концерны могут быть использованы в ресурсах, чтобы избежать дублирования кода и разделить поведение между несколькими маршрутами:

resources :messages, concerns: :commentable

resources :posts, concerns: [:commentable, :image\_attachable]

Вышеуказанное эквивалентно:

resources :messages do

resources :comments

end

resources :posts do

resources :comments

resources :images, only: :index

end

Также их можно использовать в любом месте внутри маршрутов, например в вызове scope или namespace:

namespace :posts do

concerns :commentable

end

**Создание путей и URL из объектов**

В дополнение к использованию маршрутных хелперов, Rails может также создавать пути и URL из массива параметров. Например, предположим у вас есть этот набор маршрутов:

resources :magazines do

resources :ads

end

При использовании magazine\_ad\_path, можно передать экземпляры Magazine и Ad вместо числовых ID:

<%= link\_to 'Ad details', magazine\_ad\_path(@magazine, @ad) %>

Можно также использовать url\_for с набором объектов, и Rails автоматически определит, какой маршрут вам нужен:

<%= link\_to 'Ad details', url\_for([@magazine, @ad]) %>

В этом случае Rails увидит, что @magazine это Magazine и @ad это Ad и поэтому использует хелпер magazine\_ad\_path. В хелперах, таких как link\_to, можно определить лишь объект вместо полного вызова url\_for:

<%= link\_to 'Ad details', [@magazine, @ad] %>

Если хотите ссылку только на magazine:

<%= link\_to 'Magazine details', @magazine %>

Для других экшнов следует всего лишь вставить имя экшна как первый элемент массива:

<%= link\_to 'Edit Ad', [:edit, @magazine, @ad] %>

Это позволит рассматривать экземпляры ваших моделей как URL, что является ключевым преимуществом ресурсного стиля.

**Определение дополнительных экшнов RESTful**

Вы не ограничены семью маршрутами, которые создает роутинг RESTful по умолчанию. Если хотите, можете добавить дополнительные маршруты, применяющиеся к коллекции или отдельным элементам коллекции.

**Добавление маршрутов к элементам**

Для добавления маршрута к элементу, добавьте блок member в блок ресурса:

resources :photos do

member do

get 'preview'

end

end

Это распознает /photos/1/preview с GET, и направит его в экшн preview PhotosController, со значением id ресурса, переданным в params[:id]. Это также создаст хелперы preview\_photo\_url и preview\_photo\_path.

В блоке маршрутов к элементу каждое имя маршрута определяет метод HTTP, с которым он будет распознан. Тут можно использовать get, patch, put, post или delete. Если у вас нет нескольких маршрутов к элементу, также можно передать :on к маршруту, избавившись от блока:

resources :photos do

get 'preview', on: :member

end

Можно опустить опцию :on, это создаст такой же маршрут для элемента, за исключением того, что значение id ресурса будет доступен в params[:photo\_id] вместо params[:id].

**Добавление маршрутов к коллекции**

Чтобы добавить маршрут к коллекции:

resources :photos do

collection do

get 'search'

end

end

Это позволит Rails распознать URL, такие как /photos/search с GET и направить его в экшн search PhotosController. Это также создаст маршрутные хелперы search\_photos\_url и search\_photos\_path.

Как и с маршрутами к элементу, можно передать :on к маршруту:

resources :photos do

get 'search', on: :collection

end

**Добавление маршрутов для дополнительных экшнов New**

Чтобы добавить альтернативный экшн new, используя сокращенный вариант :on:

resources :comments do

get 'preview', on: :new

end

Это позволит Rails распознавать маршруты, такие как /comments/new/preview с GET, и направлять их в экшн preview в CommentsController. Он также создаст маршрутные хелперы preview\_new\_comment\_url и preview\_new\_comment\_path.

Если вдруг вы захотели добавить много дополнительных экшнов в ресурсный маршрут, нужно остановиться и спросить себя, может быть, от вас утаилось присутствие другого ресурса.

**Нересурсные маршруты**

В дополнению к ресурсному роутингу, Rails поддерживает роутинг произвольных URL к экшнам. Тут не будет групп маршрутов, создаваемых автоматически ресурсным роутингом. Вместо этого вы должны настроить каждый маршрут вашего приложения отдельно.

Хотя обычно следует пользоваться ресурсным роутингом, есть много мест, где более подходит простой роутинг. Не стоит пытаться заворачивать каждый кусочек своего приложения в ресурсные рамки, если он плохо поддается.

В частности, простой роутинг облегчает привязку унаследованых URL к новым экшнам Rails.

**Обязательные параметры**

При настройке обычного маршрута вы предоставляете ряд символов, которые Rails связывает с частями входящего запроса HTTP. Два из этих символов специальные: :controller связывает с именем контроллера в приложении, и :action связывает с именем экшна в контроллере. Например, рассмотрим следующий маршрут:

get ':controller(/:action(/:id))'

Если входящий запрос /photos/show/1 обрабатывается этим маршрутом (так как не встретил какого-либо соответствующего маршрута в файле до этого), то результатом будет вызов экшна show в PhotosController, и результирующий параметр (1) будет доступен как params[:id]. Этот маршрут также свяжет входящий запрос /photos с PhotosController, поскольку :action и :id необязательные параметры, обозначенные скобками.

**Динамические сегменты**

Можете настроить сколько угодно динамических сегментов в обычном маршруте. Всё, кроме :controller или :action, будет доступно для соответствующего экшна как часть хэша params. Таким образом, если настроите такой маршрут:

get ':controller/:action/:id/:user\_id'

Входящий URL /photos/show/1/2 будет направлен на экшн show в PhotosController. params[:id] будет установлен как "1", и params[:user\_id] будет установлен как "2".

Нельзя использовать :namespace или :module вместе с сегментом пути :controller. Если это нужно, используйте ограничение на :controller, которое соответствует требуемому пространству имен, т.е.:

get ':controller(/:action(/:id))', controller: /admin\/[^\/]+/

По умолчанию динамические сегменты не принимают точки - потому что точка используется как разделитель для формата маршрутов. Если в динамическом сегменте необходимо использовать точку, добавьте ограничение, переопределяющее это – к примеру, id: /[^\/]+/ позволяет все, кроме слэша.

**Статичные сегменты**

Можете определить статичные сегменты при создании маршрута, не начинающиеся с двоеточия в фрагменте:

get ':controller/:action/:id/with\_user/:user\_id'

Этот маршрут соответствует путям, таким как /photos/show/1/with\_user/2. В этом случае params будет { controller: 'photos', action: 'show', id: '1', user\_id: '2' }.

**Параметры строки запроса**

params также включает любые параметры из строки запроса. Например, с таким маршрутом:

get ':controller/:action/:id'

Входящий путь /photos/show/1?user\_id=2 будет направлен на экшн show контроллера Photos. params будет { controller: 'photos', action: 'show', id: '1', user\_id: '2' }.

**Определение значений по умолчанию**

В маршруте не обязательно явно использовать символы :controller и :action. Можете предоставить их как значения по умолчанию:

get 'photos/:id', to: 'photos#show'

С этим маршрутом Rails направит входящий путь /photos/12 на экшн show в PhotosController.

Также можете определить другие значения по умолчанию в маршруте, предоставив хэш для опции :defaults. Это также относится к параметрам, которые не определены как динамические сегменты. Например:

get 'photos/:id', to: 'photos#show', defaults: { format: 'jpg' }

Rails направит photos/12 в экшн show PhotosController, и установит params[:format] как jpg.

**Именование маршрутов**

Можно определить имя для любого маршрута, используя опцию :as:

get 'exit', to: 'sessions#destroy', as: :logout

Это создаст logout\_path и logout\_url как именнованные хелперы в вашем приложении. Вызов logout\_path вернет /exit

Также это можно использовать для переопределения маршрутых методов, поределенных ресурсами, следующим образом:

get ':username', to: 'users#show', as: :user

Что определит метод user\_path, который будет доступен в контроллерах, хелперах и вьюхах, и будет вести на маршрут, такой как /bob. В экшне show из UsersController, params[:username] будет содержать имя пользователя. Измените :username в определении маршурта, если не хотите, чтобы имя параметра было :username.

**Ограничения метода HTTP**

В основном следует импользовать методы get, post, put и delete для ограничения маршрута определенным методом. Можно использовать метод match с опцией :via для соответствия нескольким методам за раз:

match 'photos', to: 'photos#show', via: [:get, :post]

Также можно установить соответствие всем методам для определенного маршрута, используя :via: :all:

match 'photos', to: 'photos#show', via: :all

Маршрутизация запросов GET и POST одновременно в один экшн небезопасна. В основном, следует избегать маршрутизацию всех методов в экшн, если у вас нет веской причины делать так.

**Ограничения сегмента**

Можно использовать опцию :constraints для соблюдения формата динамического сегмента:

get 'photos/:id', to: 'photos#show', constraints: { id: /[A-Z]\d{5}/ }

Этот маршрут соответствует путям, таким как /photos/A12345, но не /photos/893. Можно выразить более кратко тот же маршрут следующим образом:

get 'photos/:id', to: 'photos#show', id: /[A-Z]\d{5}/

:constraints принимает регулярное выражение c ограничением, что якоря regexp не могут использоваться. Например, следующий маршрут не работает:

get '/:id', to: 'posts#show', constraints: {id: /^\d/}

Однако отметьте, что нет необходимости использовать якоря, поскольку все маршруты заякорены изначально.

Например, следующие маршруты приведут к posts со значением to\_param такими как 1-hello-world, которые всегда начинаются с цифры, к users со значениями to\_param такими как david, никогда не начинающимися с цифры, разделенные в корневом пространстве имен:

get '/:id', to: 'posts#show', constraints: { id: /\d.+/ }

get '/:username', to: 'users#show'

**Ограничения, основанные на запросе**

Также можно ограничить маршрут, основываясь на любом методе в объекте Request, который возвращает String.

Ограничение, основанное на запросе, определяется так же, как и сегментное ограничение:

get 'photos', constraints: {subdomain: 'admin'}

Также можно определить ограничения в форме блока:

namespace :admin do

constraints subdomain: 'admin' do

resources :photos

end

end

**Продвинутые ограничения**

Если имеется более продвинутое ограничение, можете предоставить объект, отвечающий на matches?, который будет использовать Rails. Скажем, вы хотите направить всех пользователей через черный список в BlacklistController. Можно сделать так:

class BlacklistConstraint

def initialize

@ips = Blacklist.retrieve\_ips

end

def matches?(request)

@ips.include?(request.remote\_ip)

end

end

TwitterClone::Application.routes.draw do

get '\*path', to: 'blacklist#index',

constraints: BlacklistConstraint.new

end

Ограничения также можно определить как лямбду:

TwitterClone::Application.routes.draw do

get '\*path', to: 'blacklist#index',

constraints: lambda { |request| Blacklist.retrieve\_ips.include?(request.remote\_ip) }

end

И метод matches?, и лямбда получают объект request в качестве аргумента.

**Подстановка маршрутов и динамические сегменты**

Подстановка маршрутов - это способ указать, что определенные параметры должны соответствовать остальным частям маршрута. Например:

get 'photos/\*other', to: 'photos#unknown'

Этот маршрут будет соответствовать photos/12 или /photos/long/path/to/12, установив params[:other] как "12", или "long/path/to/12". Фрагменты, начинающиеся со звездочки, называются динамические сегменты ("wildcard segments").

Динамические сегменты могут быть где угодно в маршруте. Например:

get 'books/\*section/:title', to: 'books#show'

будет соответствовать books/some/section/last-words-a-memoir с params[:section] равным 'some/section', и params[:title] равным 'last-words-a-memoir'.

На самом деле технически маршрут может иметь более одного динамического сегмента, matcher назначает параметры интуитивным образом. Для примера:

get '\*a/foo/\*b', to: 'test#index'

будет соответствовать zoo/woo/foo/bar/baz с params[:a] равным 'zoo/woo', и params[:b] равным 'bar/baz'.

Запросив '/foo/bar.json', ваш params[:pages] будет равен 'foo/bar' с форматом запроса JSON. Если вам нужно вернуть старое поведение 3.0.x, можете предоставить format: false вот так:

get '\*pages', to: 'pages#show', format: false

Если хотите сделать сегмент формата обязательным, чтобы его нельзя было опустить, укажите format: true подобным образом:

get '\*pages', to: 'pages#show', format: true

**Перенаправление**

Можно перенаправить любой путь на другой путь, используя хелпер redirect в вашем роутере:

get '/stories', to: redirect('/posts')

Также можно повторно использовать динамические сегменты для соответствия пути, на который перенаправляем:

get '/stories/:name', to: redirect('/posts/%{name}')

Также можно предоставить блок для перенаправления, который получает символизированные параметры пути и объект request:

get '/stories/:name', to: redirect {|path\_params, req| "/posts/#{path\_params[:name].pluralize}" }

get '/stories', to: redirect {|path\_params, req| "/posts/#{req.subdomain}" }

Пожалуйста, отметьте, что это перенаправление является 301 "Moved Permanently". Учтите, что некоторые браузеры или прокси серверы закэшируют этот тип перенаправления, сделав старые страницы недоступными.

Во всех этих случаях, если не предоставить предшествующий хост (http://www.example.com), Rails возьмет эти детали из текущего запроса.

**Роутинг к приложениям Rack**

Вместо строки, подобной 'posts#index', соответствующей экшну index в PostsController, можно определить любое приложение Rack как конечную точку совпадения.

match '/application.js', to: Sprockets, via: :all

Пока Sprockets отвечает на call и возвращает [status, headers, body], роутер не будет различать приложение Rack и экшн. Здесь подходит использование via: :all, если вы хотите позволить своему приложению Rack обрабатывать все методы так, как оно считает нужным.

Для любопытства, 'posts#index' фактически расширяется до PostsController.action(:index), который возвращает валидное приложение Rack.

**Использование root**

Можно определить, с чем Rails должен связать '/' с помощью метода root:

root to: 'pages#main'

root 'pages#main' # то же самое в краткой форме

Следует поместить маршрут root в начало файла, поскольку это наиболее популярный маршрут и должен быть проверен первым.

Маршрут root связывает с экшном только запросы GET.

root также можно использовать внутри namespace и scope. Например:

namespace :admin do

root to: "admin#index"

end

root to: "home#index"

**Маршруты с символами Unicode**

Маршруты с символами unicode можно определять непосредственно. Например:

get 'こんにちは', to: 'welcome#index'

**Настройка ресурсных маршрутов**

Хотя маршруты и хелперы по умолчанию, созданные resources :posts, обычно нормально работают, вы, возможно, захотите их настроить некоторым образом. Rails позволяет настроить практически любую часть ресурсных хелперов.

**Определение используемого контроллера**

Опция :controller позволяет явно определить контроллер, используемый ресурсом. Например:

resources :photos, controller: 'images'

распознает входящие пути, начинающиеся с /photo, но смаршрутизирует к контроллеру Images:

|  |  |  |  |
| --- | --- | --- | --- |
| **Метод HTTP** | **Путь** | **Экшн** | **Именнованный хелпер** |
| GET | /photos | index | photos\_path |
| GET | /photos/new | new | new\_photo\_path |
| POST | /photos | create | photos\_path |
| GET | /photos/:id | show | photo\_path(:id) |
| GET | /photos/:id/edit | edit | edit\_photo\_path(:id) |
| PATCH/PUT | /photos/:id | update | photo\_path(:id) |
| DELETE | /photos/:id | destroy | photo\_path(:id) |

Используйте photos\_path, new\_photo\_path и т.д. для создания путей для этого ресурса.

Для контроллеров в пространстве имен можно использовать обозначение директории. Например:

resources :user\_permissions, controller: 'admin/user\_permissions'

Это будет смаршрутизировано на контроллер Admin::UserPermissions.

Поддерживается только обозначение директории. Указание контроллера с помощью обозначения константы ruby (т.е. :controller => 'Admin::UserPermissions') может привести к маршрутным проблемам, и в итоге к ошибке.

**Опеределение ограничений**

Можно использовать опцию :constraints для определения требуемого формата на неявном id. Например:

resources :photos, constraints: {id: /[A-Z][A-Z][0-9]+/}

Это объявление ограничивает параметр :id соответствием предоставленному регулярному выражению. Итак, в этом случае роутер больше не будет сопоставлять /photos/1 этому маршруту. Вместо этого будет соответствовать /photos/RR27.

Можно определить одиночное ограничение, применив его к ряду маршрутов, используя блочную форму:

constraints(id: /[A-Z][A-Z][0-9]+/) do

resources :photos

resources :accounts

end

Конечно, можете использовать более продвинутые ограничения, доступные в нересурсных маршрутах, в этом контексте

По умолчанию параметр :id не принимает точки - так как точка используется как разделитель для формата маршрута. Если необходимо использовать точку в :id, добавьте ограничение, которое переопределит это - к примеру id: /[^\/]+/ позволяет все, кроме слэша.

**Переопределение именнованных хелперов**

Опция :as позволяет переопределить нормальное именование для именнованных маршрутных хелперов. Например:

resources :photos, as: 'images'

распознает входящие пути, начинающиеся с /photos и смаршрутизирует запросы к PhotosController:

|  |  |  |  |
| --- | --- | --- | --- |
| **Метод HTTP** | **Путь** | **Экшн** | **Именнованный хелпер** |
| GET | /photos | index | images\_path |
| GET | /photos/new | new | new\_image\_path |
| POST | /photos | create | images\_path |
| GET | /photos/:id | show | image\_path(:id) |
| GET | /photos/:id/edit | edit | edit\_image\_path(:id) |
| PATCH/PUT | /photos/:id | update | image\_path(:id) |
| DELETE | /photos/:id | destroy | image\_path(:id) |

**Переопределение сегментов new и edit**

Опция :path\_names позволяет переопределить автоматически создаваемые сегменты "new" и "edit" в путях:

resources :photos, path\_names: { new: 'make', edit: 'change' }

Это приведет к тому, что роутинг распознает пути, такие как:

/photos/make

/photos/1/change

Фактические имена экшнов не меняются этой опцией. Два показанных пути все еще ведут к экшнам new и edit.

Если вдруг захотите изменить эту опцию одинаково для всех маршрутов, можно использовать scope:

scope path\_names: { new: 'make' } do

# остальные ваши маршруты

end

**Префикс именнованных маршрутных хелперов**

Можно использовать опцию :as для задания префикса именнованных маршрутных хелперов, создаваемых Rails для маршрута. Используйте эту опцию для предотвращения коллизий имен между маршрутами, используемыми пространство путей. Например:

scope 'admin' do

resources :photos, as: 'admin\_photos'

end

resources :photos

Это предоставит маршрутные хелперы такие как admin\_photos\_path, new\_admin\_photo\_path и т.д.

Для задания префикса группы маршрутов, используйте :as со scope:

scope 'admin', as: 'admin' do

resources :photos, :accounts

end

resources :photos, :accounts

Это создаст маршруты такие как admin\_photos\_path и admin\_accounts\_path, ведущие соответственно к /admin/photos и /admin/accounts.

Пространство namespace автоматически добавляет :as, так же, как и префиксы :module и :path.

Можно задать префикс маршрута именнованным параметром также так:

scope ':username' do

resources :posts

end

Это предоставит URL, такие как /bob/posts/1 и позволит обратиться к части пути username в контроллерах, хелперах и вьюхах как params[:username].

**Ограничение создаваемых маршрутов**

По умолчанию Rails создает маршруты для всех семи экшнов по умолчанию (index, show, new, create, edit, update, and destroy) для каждого маршрута RESTful вашего приложения. Можно использовать опции :only и :except для точной настройки этого поведения. Опция :only говорит Rails создать только определенные маршруты:

resources :photos, only: [:index, :show]

Теперь запрос GET к /photos будет успешным, а запрос POST к /photos (который обычно соединяется с экшном create) провалится.

Опция :except определяет маршрут или перечень маршрутов, который Rails *не* должен создавать:

resources :photos, except: :destroy

В этом случае Rails создаст все нормальные маршруты за исключением маршрута для destroy (запрос DELETE к /photos/:id).

Если в вашем приложении много маршрутов RESTful, использование :only и :except для создания только тех маршрутов, которые Вам фактически нужны, позволит снизить использование памяти и ускорить процесс роутинга.

**Переведенные пути**

Используя scope, можно изменить имена путей, создаваемых ресурсами:

scope(path\_names: { new: 'neu', edit: 'bearbeiten' }) do

resources :categories, path: 'kategorien'

end

Rails теперь создаст маршруты к CategoriesController.

|  |  |  |  |
| --- | --- | --- | --- |
| **Метод HTTP** | **Путь** | **Экшн** | **Именнованный хелпер** |
| GET | /kategorien | index | categories\_path |
| GET | /kategorien/neu | new | new\_category\_path |
| POST | /kategorien | create | categories\_path |
| GET | /kategorien/:id | show | category\_path(:id) |
| GET | /kategorien/:id/bearbeiten | edit | edit\_category\_path(:id) |
| PATCH/PUT | /kategorien/:id | update | category\_path(:id) |
| DELETE | /kategorien/:id | destroy | category\_path(:id) |

**Переопределение единственного числа**

Если хотите определить единственное число ресурса, следует добавить дополнительные правила в Inflector:

ActiveSupport::Inflector.inflections do |inflect|

inflect.irregular 'tooth', 'teeth'

end

**Использование :as во вложенных ресурсах**

Опция :as переопределяет автоматически создаваемое имя для ресурса в хелперах вложенного маршрута. Например:

resources :magazines do

resources :ads, as: 'periodical\_ads'

end

Это создаст маршрутные хелперы такие как magazine\_periodical\_ads\_url и edit\_magazine\_periodical\_ad\_path.

**Осмотр и тестирование маршрутов**

Rails предлагает инструменты для осмотра и тестирования маршрутов.

**Список существующих маршрутов**

Чтобы получить полный список всех доступных маршрутов вашего приложения, посетите http://localhost:3000/rails/info/routes в браузере, в то время как ваш сервер запущен в режиме **development**. Команда rake routes, запущенная в терминале, выдаст тот же результат.

Оба метода напечатают все ваши маршруты, в том же порядке, что они появляются в routes.rb. Для каждого маршрута вы увидите:

* Имя маршрута (если имеется)
* Используемый метод HTTP (если маршрут реагирует не на все методы)
* Шаблон URL
* Параметры роутинга для этого маршрута

Например, вот небольшая часть результата команды rake routes для маршрута RESTful:

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

Можете ограничить перечень маршрутами, ведущими к определенному контроллеру, установкой переменной среды CONTROLLER:

$ CONTROLLER=users rake routes

Результат команды rake routes более читаемый, если у вас в окне терминала прокрутка, а не перенос строк.

**Тестирование маршрутов**

Маршруты должны быть включены в вашу стратегию тестирования (так же, как и остальное в вашем приложении). Rails предлагает три "встроенных оператора контроля":<http://api.rubyonrails.org/classes/ActionController/Assertions/RoutingAssertions.html>, разработанных для того, чтобы сделать тестирование маршрутов проще:

* assert\_generates
* assert\_recognizes
* assert\_routing

**Оператор контроля assert\_generates**

Используйте assert\_generates чтобы убедиться в том, что определенный набор опций создает конкретный путь. Можете использовать его с маршрутами по умолчанию или своими маршрутами. Например:

assert\_generates '/photos/1', { controller: 'photos', action: 'show', id: '1' }

assert\_generates '/about', controller: 'pages', action: 'about'

**Оператор контроля assert\_recognizes**

Оператор контроля assert\_recognizes - это противоположность assert\_generates. Он убеждается, что Rails распознает предложенный путь и маршрутизирует его в конкретную точку в вашем приложении. Например:

assert\_recognizes({ controller: 'photos', action: 'show', id: '1' }, '/photos/1')

Можете задать аргумент :method, чтобы определить метод HTTP:

assert\_recognizes({ controller: 'photos', action: 'create' }, { path: 'photos', method: :post })

**Оператор контроля assert\_routing**

Оператор контроля assert\_routing проверяет маршрут с двух сторон: он тестирует, что путь генерирует опции, и что опции генерируют путь. Таким образом, он комбинирует функции assert\_generates и assert\_recognizes:

assert\_routing({ path: 'photos', method: :post }, { controller: 'photos', action: 'create' })