**Разбираемся с простыми формами**

Самый простой хелпер форм - это form\_tag.

|  |
| --- |
| <%= form\_tag do %>    Содержимое формы  <% end %> |

При подобном вызове без аргументов, он создает тег <form>, который, при отправке, сформирует POST-запрос к текущей странице. Например, предположим текущая страница /home/index, тогда сгенерированный HTML будет выглядеть так (некоторые переводы строк добавлены для читаемости):

|  |
| --- |
| <form accept-charset="UTF-8" action="/home/index" method="post">    <div style="margin:0;padding:0">      <input name="utf8" type="hidden" value="&#x2713;" />      <input name="authenticity\_token" type="hidden" value="f755bb0ed134b76c432144748a6d4b7a7ddf2b71" />    </div>    Form contents  </form> |

Можно увидеть, что HTML содержит нечто дополнительное: элемент div с двумя скрытыми input внутри. Этот div важен, поскольку без него форма не может быть успешно отправлена. Первый элемент input с именем utf8 обеспечивает, чтобы браузер правильно относился к кодировке вашей формы, он генерируется для всех форм, у которых action равен "GET" или "POST". Второй элемент input с именем authenticity\_token является особенностью безопасности Rails, называемой **защитой от подделки межсайтовых запросов**, и хелперы форм генерируют его для каждой формы, у которых action не "GET" (если эта особенность безопасности включена). Подробнее об этом можно прочитать в Руководстве Ruby On Rails по безопасности.

На всем протяжении этого руководства, div со скрытыми элементами input будет исключаться для краткости.

**Характерная форма поиска**

Одной из наиболее простых форм, встречающихся в вебе, является форма поиска. Эта форма содержит:

* элемент формы с методом "GET",
* метку для поля ввода,
* элемент поля ввода текста и
* элемент отправки.

Чтобы создать эту форму, используем, соответственно, form\_tag, label\_tag, text\_field\_tag и submit\_tag. Как здесь:

|  |
| --- |
| <%= form\_tag("/search", method: "get") do %>    <%= label\_tag(:q, "Search for:") %>    <%= text\_field\_tag(:q) %>    <%= submit\_tag("Search") %>  <% end %> |

Это создаст следующий HTML:

|  |
| --- |
| <form accept-charset="UTF-8" action="/search" method="get">    <label for="q">Search for:</label>    <input id="q" name="q" type="text" />    <input name="commit" type="submit" value="Search" />  </form> |

Для каждого поля формы генерируется атрибут ID из его имени ("q" в примере). Эти ID могут быть очень полезны для стилей CSS или управления полями форм с помощью JavaScript.

Кроме text\_field\_tag и submit\_tag имеется похожий хелпер для *каждого* элемента управления формой в HTML.

Всегда используйте "GET" как метод для форм поиска. Это позволит пользователям сохранить в закладки определенный поиск и потом вернуться к нему. В более общем плане Rails призывает вас использовать правильный метод HTTP для экшна.

**Несколько хэшей в вызовах хелпера формы**

Хелпер form\_tag принимает 2 аргумента: путь для экшна и хэш опций. Этот хэш определяет метод отправки формы и опции HTML, такие как класс элемента form.

Как в случае с хелпером link\_to, аргумент пути не должен быть строкой. Это может быть хэш параметров URL, распознаваемый механизмом маршрутизации Rails, преобразующим хэш в валидный URL. Однако, если оба аргумента для form\_tag хэши, можно легко получить проблему, если захотите определить оба. Например, вы напишите так:

|  |
| --- |
| form\_tag(controller: "people", action: "search", method: "get", class: "nifty\_form")  # => '<form accept-charset="UTF-8" action="/people/search?method=get&class=nifty\_form" method="post">' |

Здесь методы method и class присоединены к строке запроса сгенерированного URL, поскольку, хотя вы и хотели передать два хэша, фактически вы передали один. Чтобы сообщить об этом Ruby, следует выделить первый хэш (или оба хэша) фигурными скобками. Это создаст HTML, который вы ожидаете:

|  |
| --- |
| form\_tag({controller: "people", action: "search"}, method: "get", class: "nifty\_form")  # => '<form accept-charset="UTF-8" action="/people/search" method="get" class="nifty\_form">' |

**Хелперы для создания элементов форм**

Rails предоставляет ряд хелперов для генерации элементов форм, таких как чекбоксы, текстовые поля, радио-кнопки и так далее. Эти простые хелперы с именами, оканчивающимися на "\_tag" (такие как text\_field\_tag и check\_box\_tag), генерируют отдельный элемент <input>. Первый параметр у них это всегда имя поля. Когда форма отправлена, имя будет передано среди данных формы, и, в свою очередь, помещено в хэш params со значением, введенным пользователем в это поле. Например, если форма содержит <%= text\_field\_tag(:query) %>, то значение для этого поля можно получить в контроллере с помощью params[:query].

При именовании полей Rails использует определенные соглашения, делающие возможным отправлять параметры с нескалярными величинами, такие как массивы и хэши, которые также будут доступны в params.

**Чекбоксы**

Чекбоксы - это элементы управления формой, которые дают пользователю ряд опций, которые он может включить или выключить:

|  |
| --- |
| <%= check\_box\_tag(:pet\_dog) %>  <%= label\_tag(:pet\_dog, "I own a dog") %>  <%= check\_box\_tag(:pet\_cat) %>  <%= label\_tag(:pet\_cat, "I own a cat") %> |

Это создаст следующее:

|  |
| --- |
| <input id="pet\_dog" name="pet\_dog" type="checkbox" value="1" />  <label for="pet\_dog">I own a dog</label>  <input id="pet\_cat" name="pet\_cat" type="checkbox" value="1" />  <label for="pet\_cat">I own a cat</label> |

Первый параметр у check\_box\_tag, разумеется, - это имя поля. Второй параметр, естественно, - это значение поля. Это значение будет включено в данные формы (и будет присутствовать в params), когда чекбокс нажат.

**Радио-кнопки**

Радио-кнопки, чем-то похожие на чекбоксы, - это элементы управления, которые определяют набор взаимоисключающих опций (т.е. пользователь может выбрать только одну):

|  |
| --- |
| <%= radio\_button\_tag(:age, "child") %>  <%= label\_tag(:age\_child, "I am younger than 21") %>  <%= radio\_button\_tag(:age, "adult") %>  <%= label\_tag(:age\_adult, "I'm over 21") %> |

Результат:

|  |
| --- |
| <input id="age\_child" name="age" type="radio" value="child" />  <label for="age\_child">I am younger than 21</label>  <input id="age\_adult" name="age" type="radio" value="adult" />  <label for="age\_adult">I'm over 21</label> |

Как и у check\_box\_tag, второй параметр для radio\_button\_tag это значение поля. Так как эти две радио-кнопки имеют одинаковое имя (age), пользователь может выбрать одну, и params[:age] будет содержать или "child" или "adult".

Всегда используйте метки (labels) для чекбоксов и радио-кнопок. Они связывают текст с определенной опцией и, предоставляя большее пространство для щелчка, упрощают ввод.

**Другие интересные хелперы**

Среди других элементов управления формой стоит упомянуть текстовое поле, поле пароля, скрытое поле, поля поиска, ввода телефона, даты, времени, цвета, даты-времени, локальных даты-времени, месяца, недели, url и email:

|  |
| --- |
| <%= text\_area\_tag(:message, "Hi, nice site", size: "24x6") %>  <%= password\_field\_tag(:password) %>  <%= hidden\_field\_tag(:parent\_id, "5") %>  <%= search\_field(:user, :name) %>  <%= telephone\_field(:user, :phone) %>  <%= date\_field(:user, :born\_on) %>  <%= datetime\_field(:user, :meeting\_time) %>  <%= datetime\_local\_field(:user, :graduation\_day) %>  <%= month\_field(:user, :birthday\_month) %>  <%= week\_field(:user, :birthday\_week) %>  <%= url\_field(:user, :homepage) %>  <%= email\_field(:user, :address) %>  <%= color\_field(:user, :favorite\_color) %>  <%= time\_field(:task, :started\_at) %> |

результат:

|  |
| --- |
| <textarea id="message" name="message" cols="24" rows="6">Hi, nice site</textarea>  <input id="password" name="password" type="password" />  <input id="parent\_id" name="parent\_id" type="hidden" value="5" />  <input id="user\_name" name="user[name]" type="search" />  <input id="user\_phone" name="user[phone]" type="tel" />  <input id="user\_born\_on" name="user[born\_on]" type="date" />  <input id="user\_meeting\_time" name="user[meeting\_time]" type="datetime" />  <input id="user\_graduation\_day" name="user[graduation\_day]" type="datetime-local" />  <input id="user\_birthday\_month" name="user[birthday\_month]" type="month" />  <input id="user\_birthday\_week" name="user[birthday\_week]" type="week" />  <input id="user\_homepage" name="user[homepage]" type="url" />  <input id="user\_address" name="user[address]" type="email" />  <input id="user\_favorite\_color" name="user[favorite\_color]" type="color" value="#000000" />  <input id="task\_started\_at" name="task[started\_at]" type="time" /> |

Скрытые поля не отображаются пользователю, вместо этого они содержат данные, как и любое текстовое поле. Их значения могут быть изменены с помощью JavaScript.

Поля поиска, ввода телефона, даты, времени, цвета, даты-времени, локальных даты-времени, месяца, недели, url и email - это элементы управления HTML5. Если необходимо, чтобы у вашего приложения была совместимость со старыми браузерами, вам необходим HTML5 polyfill (предоставляемый с помощью CSS и/или JavaScript). Хотя в таких решениях [нет недостатка](https://github.com/Modernizr/Modernizr/wiki/HTML5-Cross-Browser-Polyfills), популярными инструментами на сегодняшний момент являются [Modernizr](http://www.modernizr.com/) и [yepnope](http://yepnopejs.com/), предоставляющие простой способ добавить функциональность, основанную на присутствии обнаруженных особенностей HTML5.

**Работаем с объектами модели**

**Хелперы объекта модели**

Наиболее частыми задачами для форм являются редактирование или создание объекта модели. В то время как хелперы \*\_tag, конечно, могут быть использованы для этой задачи, они несколько многословны, так как для каждого тега вам придется обеспечить использование правильного имени параметра и установку подходящего значения поля по умолчанию. Rails предоставляет методы, подогнанные под эту задачу. У этих хелперов отсутствует суффикс *tag, например, `text*field,text\_area`.

У этих хелперов первый аргумент - это имя переменной экземпляра, а второй - это имя метода (обычно атрибутного), вызываемого для этого объекта. Rails установит значение элемента управления равным возвращаемому значению метода объекта и установит подходящее имя поля. Если ваш контроллер определил @person и имя этой персоны Henry, тогда форма, содержащая:

|  |
| --- |
| <%= text\_field(:person, :name) %> |

выдаст подобный результат

|  |
| --- |
| <input id="person\_name" name="person[name]" type="text" value="Henry"/> |

После подтверждения формы, значение, введенное пользователем, будет храниться в params[:person][:name]. Хэш params[:person] годен для передачи в Person.new или, если @person - это экземпляр Person, в @person.update. Хотя имя атрибута - очень распространенный второй параметр для этих хелперов, он не является обязательным. В вышеупомянутом примере, до тех пор пока объекты person имеют методы name и name=, Rails будет удовлетворен.

Необходимо передавать имя переменной экземпляра, т.е. :person или "person", а не фактический экземпляр объекта вашей модели.

Rails предоставляет хелперы для отображения ошибок валидации, связанных с объектом модели. Детально они раскрываются в руководстве Отображение ошибок валидации во вьюхе.

**Привязывание формы к объекту**

Хотя комфортность несколько улучшилась, она еще далека от совершенства. Если у Person много атрибутов для редактирования, тогда мы должны повторить имя редактируемого объекта много раз. То, что мы хотим сделать, - это как-то привязать форму к объекту модели, что как раз осуществляется с помощью form\_for.

Допустим у нас есть контроллер для работы со статьями articles\_controller.rb:

|  |
| --- |
| def new    @article = Article.new  end |

Соответствующая вьюха articles/new.html.erb:, использующая form\_for, выглядит так

|  |
| --- |
| <%= form\_for @article, url: {action: "create"}, html: {class: "nifty\_form"} do |f| %>    <%= f.text\_field :title %>    <%= f.text\_area :body, size: "60x12" %>    <%= f.submit "Create" %>  <% end %> |

Следует отметить несколько вещей:

* @article - это фактический объект, который редактируется.
* Здесь есть одиночный хэш опций. Опции маршрутизации передаются в хэше :url, опции HTML передаются в хэше :html. Также для формы можно предоставить опцию :namespace, чтобы быть уверенным в уникальности аттрибутов id элементов формы. Аттрибут namespace будет префиксом с подчеркиванием в генерируемых для HTML id.
* Метод form\_for предоставляет объект **form builder** (переменная f).
* Методы создания элементов управления формой вызываются *для* объекта form builder f

Итоговый HTML:

|  |
| --- |
| <form accept-charset="UTF-8" action="/articles/create" method="post" class="nifty\_form">    <input id="article\_title" name="article[title]" type="text" />    <textarea id="article\_body" name="article[body]" cols="60" rows="12"></textarea>    <input name="commit" type="submit" value="Create" />  </form> |

Имя, переданное в form\_for, контролирует ключ, используемый в params для доступа к значениям формы. В примере имя article, и, таким образом, все поля формы имеют имена article[attribute\_name]. Соответственно, в экшне create хэш params[:article] имеет ключи :title и :body. О значимости имен полей ввода подробнее можно прочитать в разделе про имена параметров.

Методы хелпера, вызываемые из form builder, идентичны хелперам объекта модели, за исключением того, что не нужно указывать, какой объект будет редактироваться, так как это уже регулируется в form builder.

Можно создать подобное привязывание без фактического создания тега <form> с помощью хелпера fields\_for. Это полезно для редактирования дополнительных объектов модели в той же форме. Например, если имеем модель Person со связанной моделью ContactDetail, Вы можете создать форму для создания обеих моделей подобным образом:

|  |
| --- |
| <%= form\_for @person, url: {action: "create"} do |person\_form| %>    <%= person\_form.text\_field :name %>    <%= fields\_for @person.contact\_detail do |contact\_details\_form| %>      <%= contact\_details\_form.text\_field :phone\_number %>    <% end %>  <% end %> |

которая выдаст такой результат:

|  |
| --- |
| <form accept-charset="UTF-8" action="/people/create" class="new\_person" id="new\_person" method="post">    <input id="person\_name" name="person[name]" type="text" />    <input id="contact\_detail\_phone\_number" name="contact\_detail[phone\_number]" type="text" />  </form> |

Объект, предоставляемый fields\_for, - это form builder, подобный тому, который предоставляется form\_for (фактически form\_for внутри себя вызывает fields\_for).

**Положитесь на идентификацию записи**

Модель Article непосредственно доступна пользователям приложения, и таким образом, следуя лучшим рекомендациям разработки на Rails, вы должны объявить ее как **ресурс**.

|  |
| --- |
| resources :articles |

Объявление ресурса имеет несколько побочных эффектов. Смотрите Роутинг в Rails для подробностей по настройке и использованию ресурсов.

Когда работаем с ресурсами RESTful, вызовы form\_for могут стать значительно проще, если их основывать на **идентификации записи**. Вкратце, вы должны всего лишь передать экземпляр модели и позволить Rails выяснить имя модели и остальное:

|  |
| --- |
| ## Создание новой статьи  # длинный стиль:  form\_for(@article, url: articles\_path)  # то же самое, короткий стиль (используется идентификация записи):  form\_for(@article)    ## Редактирование существующей статьи  # длинный стиль:  form\_for(@article, url: article\_path(@article), html: {method: "patch"})  # короткий стиль:  form\_for(@article) |

Отметьте, как вызов короткого стиля form\_for остается тем же самым, независимо от того, будет запись новой или существующей. Идентификация записи достаточно сообразительная, чтобы выяснить, новая ли запись, запрашивая record.new\_record?. Она также выбирает правильный путь для подтверждения и имя, основывающееся на классе объекта.

Rails также автоматически установит надлежащие атрибуты формы class и id: форма, создающая статью, будет иметь id и class new\_article. Если редактируется статья с id=23, то class будет установлен как edit\_article, и id - как edit\_article\_23. Эти атрибуты будут опускаться для краткости далее в этом руководстве.

Когда используется STI (single-table inheritance, наследование с единой таблицей) с вашими моделями, нельзя полагаться на идентификацию записей субкласса, если лишь родительский класс определен ресурсом. Вы должны определить имя модели, :url и :method явно.

**Работаем с пространствами имен**

Если вы создали пространство имен маршрутов, form\_for также можно изящно сократить. Если у приложения есть пространство имен admin, то

|  |
| --- |
| form\_for [:admin, @article] |

создаст форму, которая передается контроллеру статей в пространстве имен admin (передача в admin\_article\_path(@article) в случае с обновлением). Если у вас несколько уровней пространства имен, тогда синтаксис подобный:

|  |
| --- |
| form\_for [:admin, :management, @article] |

Более подробно о системе маршрутизации Rails и связанным соглашениям смотрите Роутинг в Rails.

**Как формы работают с методами PATCH, PUT или DELETE?**

Фреймворк Rails поддерживает дизайн RESTful в ваших приложениях, что означает частое использование запросов "PATCH" и "DELETE" (помимо "GET" и "POST"). Однако, большинство браузеров *не поддерживают* методы, иные, чем "GET" и "POST", когда они исходят от подтверждаемых форм.

Rails работает с этой проблемой, эмулируя другие методы с помощью POST со скрытым полем, названным "\_method", который установлен для отражения желаемого метода:

|  |
| --- |
| form\_tag(search\_path, method: "patch") |

результат:

|  |
| --- |
| <form accept-charset="UTF-8" action="/search" method="post">    <div style="margin:0;padding:0">      <input name="\_method" type="hidden" value="patch" />      <input name="utf8" type="hidden" value="&#x2713;" />      <input name="authenticity\_token" type="hidden" value="f755bb0ed134b76c432144748a6d4b7a7ddf2b71" />    </div>    ... |

При парсинге данных POST, Rails принимает в счет специальный параметр \_method и действует с ним, как будто бы был определен этот метод HTTP ("PATCH" в этом примере).

**Легкое создание списков выбора**

Списки выбора в HTML требуют значительной верстки (один элемент OPTION для каждого варианта выбора), поэтому имеет большой смысл создавать их динамически.

Вот как может выглядеть верстка:

|  |
| --- |
| <select name="city\_id" id="city\_id">    <option value="1">Lisbon</option>    <option value="2">Madrid</option>    ...    <option value="12">Berlin</option>  </select> |

Тут мы имеем перечень городов, имена которых представлены пользователю. Самому приложению для обработки нужен только их ID, поэтому он используется как атрибут value варианта выбора. Давайте посмотрим, как Rails может нам помочь.

h4. Тэги Select и Option

Наиболее простой хелпер - это select\_tag, который, как следует из имени, просто создает тег SELECT, инкапсулирующий строку опций:

|  |
| --- |
| <%= select\_tag(:city\_id, '<option value="1">Lisbon</option>...') %> |

Это закладывает начало, но пока еще динамически не создает теги вариантов выбора. Вы можете создать теги option с помощью хелпера options\_for\_select:

|  |
| --- |
| <%= options\_for\_select([['Lisbon', 1], ['Madrid', 2], ...]) %>    результат:    <option value="1">Lisbon</option>  <option value="2">Madrid</option>  ... |

Первый аргумент для options\_for\_select - это вложенный массив, в котором каждый элемент содержит два элемента: текст варианта (название города) и значение варианта (id города). Значение варианта - это то, что будет передано в ваш контроллер. Часто бывает, что значение - это id соответствующего объекта базы данных, но это не всегда так.

Зная это, вы можете комбинировать select\_tag и options\_for\_select для достижения желаемой полной верстки:

|  |
| --- |
| <%= select\_tag(:city\_id, options\_for\_select(...)) %> |

options\_for\_select позволяет вам предварительно выбрать вариант, передав его значение.

|  |
| --- |
| <%= options\_for\_select([['Lisbon', 1], ['Madrid', 2], ...], 2) %>    результат:    <option value="1">Lisbon</option>  <option value="2" selected="selected">Madrid</option>  ... |

Всякий раз, когда Rails видит внутреннее значение создаваемого варианта, соответствующего этому значению, он добавит атрибут selected к нему.

Второй аргумент для options\_for\_select должен быть точно равен желаемому внутреннему значению. В частности, если значение - число 2, вы не можете передать "2" в options\_for\_select - вы должны передать 2. Имейте это в виду при использовании значений, извлеченных из хэша params, так как они всегда строчные.

Когда отсутствует :include\_blank или :prompt, :include\_blank становится true, если атрибут required - true, отображаемый size - 1 и multiple не true.

С помощью хэшей можно добавить произвольные атрибуты в option:

|  |
| --- |
| <%= options\_for\_select([['Lisbon', 1, {'data-size' => '2.8 million'}], ['Madrid', 2, {'data-size' => '3.2 million'}]], 2) %>    результат:    <option value="1" data-size="2.8 million">Lisbon</option>  <option value="2" selected="selected" data-size="3.2 million">Madrid</option>  ... |

**Списки выбора для работы с моделями**

В большинстве случаев элементы управления формой будут связаны с определенной моделью базы данных, и, как вы наверное и ожидали, Rails предоставляет хелперы, предназначенные для этой цели. Как в случае с другими хелперами форм, когда работаете с моделями, суффикс \_tag отбрасывается от select\_tag:

|  |
| --- |
| # контроллер:  @person = Person.new(city\_id: 2) |
| # вьюха:  <%= select(:person, :city\_id, [['Lisbon', 1], ['Madrid', 2], ...]) %> |

Отметьте, что третий параметр - массив опций - имеет тот же самый тип аргумента, что мы передавали в options\_for\_select. Преимущество в том, что не стоит беспокоиться об предварительном выборе правильного города, если пользователь уже выбрал его - Rails сделает это за вас, прочитав из атрибута @person.city\_id.

Как и в других хелперах, если хотите использовать хелпер select в form builder с областью видимостью объекта @person, синтаксис будет такой:

|  |
| --- |
| # select в form builder  <%= f.select(:city\_id, ...) %> |

При использовании select (или подобного хелпера, такого как collection\_select, select\_tag), чтобы установить связь belongs\_to, вы должны передасть имя внешнего ключа (в примере выше city\_id), а не само имя связи. Если определите city вместо city\_id, Active Record вызовет ошибку в строке ActiveRecord::AssociationTypeMismatch: City(#17815740) expected, got String(#1138750), когда вы передадите хэш params в Person.new или update. Можно взглянуть на это по другому, что хелперы форм редактируют только атрибуты. Также вам стоит пеобеспокоиться о потенциальных последствиях безопасности, если разрешить пользователям редактировать внешние ключи напрямую.

**Тэги варианта выбора из коллекции произвольных объектов**

Создание тегов вариантов с помощью options\_for\_select требует, чтобы вы создали массив, содержащий текст и значение для каждого варианта. Но что, если мы имеем модель City (вероятно даже модель Active Record) и хотим создать теги вариантов из коллекции этих объектов? Одно из решений будет сделать вложенный массив с помощью итераций:

|  |
| --- |
| <% cities\_array = City.all.map { |city| [city.name, city.id] } %>  <%= options\_for\_select(cities\_array) %> |

Хотя это и валидное решение, но Rails предоставляет менее многословную альтернативу: options\_from\_collection\_for\_select. Этот хелпер принимает коллекцию произвольных объектов и два дополнительных аргумента - имена методов для считывания опций **value** и **text**, соответственно:

|  |
| --- |
| <%= options\_from\_collection\_for\_select(City.all, :id, :name) %> |

Как следует из имени, это генерирует только теги option. Для генерации работающего списка выбора его необходимо использовать в сочетании с select\_tag, как это делалось для options\_for\_select. Когда работаем с объектами модели, так же, как select комбинирует select\_tag и options\_for\_select, collection\_select комбинирует select\_tag с options\_from\_collection\_for\_select.

|  |
| --- |
| <%= collection\_select(:person, :city\_id, City.all, :id, :name) %> |

Напомним, что options\_from\_collection\_for\_select в collection\_select - то же самое, что и options\_for\_select в select.

Пары, переданные в options\_for\_select должны сперва иметь имя, затем id, однако для options\_from\_collection\_for\_select первый аргумент - это метод значения, а второй аргумент - метод текста.

**Выбор часового пояса и страны**

Для управления поддержкой часовых поясов в Rails, можете спрашивать своих пользователей, в какой зоне они находятся. Это потребует создать варианты выбора из списка предопределенных объектов TimeZone, используя collection\_select, но вы можете просто использовать хелпер time\_zone\_select, который уже все это содержит:

|  |
| --- |
| <%= time\_zone\_select(:person, :time\_zone) %> |

Также есть хелпер time\_zone\_options\_for\_select для ручного (и поэтому гибко настраиваемого) способа осуществления этого. Читайте документацию по API, чтобы узнать о доступных аргументах для этих двух методов.

Rails *раньше* имел хелпер country\_select для выбора стран, но сейчас он вынесен во внешний [плагин country\_select](https://github.com/stefanpenner/country_select). При его использовании убедитесь, что включение или исключение определенных имен из списка может быть несколько спорным (это и послужило причиной извлечения этой функциональности из Rails).

**Использование хелперов даты и времени**

Можно выбрать не использовать хелперы форм, создающие поля ввода даты и времени HTML5, а использовать альтернативные хелперы даты и времени. Эти хелперы даты и времени отличаются от остальных хелперов форм в двух важных аспектах:

* Дата и время не представлены отдельным элементом ввода. Вместо них есть несколько, один на каждый компонент (год, месяц, день и т.д.), и, таким образом, нет одного значения в хэше params с вашими датой и временем.
* Другие хелперы используют суффикс \_tag для обозначения, является ли хелпер скелетным, либо работает на основе объектов модели. Что касается дат и времени, select\_date, select\_time и select\_datetime - это скелетные хелперы, а date\_select, time\_select и datetime\_select - это эквивалентные хелперы объекта модели.

Оба эти семейства хелперов создадут ряд списков выбора для различных компонент (год, месяц, день и т.д.).

**Скелетные хелперы**

Семейство хелперов select\_\* принимает как первый аргумент экземпляр Date, Time или DateTime, который используется как текущее выбранное значение. Можете опустить этот параметр в случае, если используется текущая дата. Например

|  |
| --- |
| <%= select\_date Date.today, prefix: :start\_date %> |

выведет (с опущенными для краткости начальными значениями вариантов)

|  |
| --- |
| <select id="start\_date\_year" name="start\_date[year]"> ... </select>  <select id="start\_date\_month" name="start\_date[month]"> ... </select>  <select id="start\_date\_day" name="start\_date[day]"> ... </select> |

Эти элементы ввода выдадут результат в params[:start\_date], являющийся хэшем с ключами :year, :month, :day. Чтобы получить фактический объект Time или Date, необходимо извлечь эти значения и передать их в подходящий конструктор, например

|  |
| --- |
| Date.civil(params[:start\_date][:year].to\_i, params[:start\_date][:month].to\_i, params[:start\_date][:day].to\_i) |

Опция :prefix это ключ, используемый для получения хэша компонентов даты из хэша params. Здесь она была установлена как start\_date, если опущена, то по умолчанию равна date.

**Хелперы объекта модели**

select\_date не очень хорошо работает с формами, обновляющими или создающими объекты Active Record, так как Active Record ожидает, что каждый элемент хэша params соответствует одному атрибуту.

Хелперы объекта модели для даты и времени возвращает параметры со специальными именами, и когда Active Record видит параметры с такими именами, он знает, что они должны быть скомбинированы с другими параметрами, и передает конструктору подходящее значения для типа столбца. Например:

|  |
| --- |
| <%= date\_select :person, :birth\_date %> |

выдаст (с опущенными для краткости начальными значениями вариантов)

|  |
| --- |
| <select id="person\_birth\_date\_1i" name="person[birth\_date(1i)]"> ... </select>  <select id="person\_birth\_date\_2i" name="person[birth\_date(2i)]"> ... </select>  <select id="person\_birth\_date\_3i" name="person[birth\_date(3i)]"> ... </select> |

что приведет к такому результату в хэше params

|  |
| --- |
| {'person' => {'birth\_date(1i)' => '2008', 'birth\_date(2i)' => '11', 'birth\_date(3i)' => '22'}} |

Когда это передастся в Person.new (или update), Active Record отметит, что эти параметры должны быть использованы, для конструирования атрибута birth\_date и использует суффиксную информацию для определения, в каком порядке должен передать эти параметры в функции, такие как Date.civil.

**Общие опции**

Оба семейства хелперов используют одинаковый базовый набор функций для создания индивидуальных тегов select, таким образом, они оба принимают множество одинаковых опций. В частности, по умолчанию Rails создаст варианты выбора года как текущий год плюс/минус пять лет. Если это неподходящий вариант, опции :start\_year и :end\_year переопределяют это. Для получения исчерпывающего перечня доступных опций обратитесь к [документации по API](http://api.rubyonrails.org/classes/ActionView/Helpers/DateHelper.html).

Как правило, следует использовать date\_select при работе с объектами модели и select\_date в иных случаях, таких как формы поиска, в которых результаты фильтруются по дате.

В большинстве случаев встроенные элементы подбора дат неуклюжи в использовании, так как не позволяют пользователю видеть отношения между датой и днями недели.

**Индивидуальные компоненты**

Иногда необходимо отобразить лишь одиночный компонент даты, такой как год или месяц. Rails предоставляет ряд хелперов для этого, по одному для каждого компонента select\_year, select\_month, select\_day, select\_hour, select\_minute, select\_second. Эти хелперы достаточно простые. По умолчанию они создадут поле ввода, названное по имени компонента времени (например "year" для select\_year, "month" для select\_month и т.д.), хотя это может быть переопределено в опции :field\_name. Опция :prefix работает так же, как работает для select\_date и select\_time и имеет такое же значение по умолчанию.

Первый параметр определяет значение даты, которое будет предложено для выбора (соответствующий элемнт может быть извлечен из экземпляра Date, Time или DateTime), либо сразу задается числовым значением. Например

|  |
| --- |
| <%= select\_year(2011) %>  <%= select\_year(Time.now) %> |

создаст такой же результат, если сейчас 2011 год, и значение, выбранное пользователем, может быть получено как params[:date][:year].

**Загрузка файлов**

Частой задачей является загрузка некоторого файла, или аватарки, или файла CSV, содержащего информацию для обработки. Самая важная вещь, это помнить при загрузке файла, что кодирование формы *ДОЛЖНО* быть установлено как "multipart/form-data". Если используете form\_for, это будет выполнено автоматически. Если используете form\_tag, нужно установить это самому, как в следующем примере.

Следующие две формы обе загружают файл.

|  |
| --- |
| <%= form\_tag({action: :upload}, multipart: true) do %>    <%= file\_field\_tag 'picture' %>  <% end %>    <%= form\_for @person do |f| %>    <%= f.file\_field :picture %>  <% end %> |

Rails предоставляет обычную пару хелперов: скелетный file\_field\_tag и модельно-ориентированный file\_field. Единственное отличие от других хелперов в том, что нельзя установить значение по умолчанию для поля ввода файла, так как в этом нет смысла. Как и следует ожидать, в первом случае загруженный файл находится в params[:picture], а во втором случае в params[:person][:picture].

**Что имеем загруженным**

Объект в хэше params - это экземпляр одного из подклассов класса IO. В зависимости от размера загруженного файла, это может оказаться либо экземпляр класса StringIO, либо экземпляр класса File, отраженный на временный файл в папке временных файлов. В обоих случаях объект будет иметь атрибут original\_filename, содержащий имя файла на компьютере пользователя, и атрибут content\_type, содержащий тип MIME загруженного файла. Следующий отрывок сохраняет загруженное содержимое в #{Rails.root}/public/uploads под тем же именем, что и оригинальный файл (предположив, что форма была одна из предыдущего примера).

|  |
| --- |
| def upload    uploaded\_io = params[:person][:picture]    File.open(Rails.root.join('public', 'uploads', uploaded\_io.original\_filename), 'wb') do |file|      file.write(uploaded\_io.read)    end  end |

Как только файл был загружен, появляется множество потенциальных задач, начиная от того, где хранить файлы (на диске, Amazon S3 и т.д.), и как связать их с моделями, до изменения размера файлов изображений и создания эскизов. Такие частности выходят за рамки данного руководства, но имеется несколько библиотек, разработанных для содействия этому. Две лучших из них - это [CarrierWave](https://github.com/jnicklas/carrierwave) и [Paperclip](http://www.thoughtbot.com/projects/paperclip).

Если пользователь не выбрал файл, соответствующий параметр останется пустой строкой.

**Работа с Ajax**

В отличие от других форм, форма асинхронной загрузки файла - это не просто передача параметра remote: true в form\_for. В Ajax-форме сериализация происходит посредством JavaScript, исполняющимся внутри браузера, и, поскольку JavaScript не может прочесть файлы с жесткого диска, файл не может быть загружен. Наиболее частым решением является использование невидимого iframe, который служит целью для отправки формы.

**Настройка Form Builder**

Как ранее упоминалось, объект, который передается от form\_for и fields\_for, - это экземпляр FormBuilder (или его подкласса). Form builder инкапсулирует представление элементов формы для отдельного объекта. Хотя, конечно, можно писать хелперы для своих форм обычным способом, вы также можете объявить подкласс FormBuilder и добавить хелперы туда. Например

|  |
| --- |
| <%= form\_for @person do |f| %>    <%= text\_field\_with\_label f, :first\_name %>  <% end %> |

может быть заменено этим

|  |
| --- |
| <%= form\_for @person, builder: LabellingFormBuilder do |f| %>    <%= f.text\_field :first\_name %>  <% end %> |

через определение класса LabellingFormBuilder подобным образом:

|  |
| --- |
| class LabellingFormBuilder < ActionView::Helpers::FormBuilder    def text\_field(attribute, options={})      label(attribute) + super    end  end |

Если это используется часто, можно определить хелпер labeled\_form\_for который автоматически определяет опцию builder: LabellingFormBuilder.

Form builder также определяет, что произойдет, если вы сделаете

|  |
| --- |
| <%= render partial: f %> |

Если f - это экземпляр FormBuilder, тогда это отрендерит партиал form, установив объект партиала как form builder. Если form builder класса LabellingFormBuilder, тогда вместо этого будет отрендерен партиал labelling\_form.

**Понимание соглашений по именованию параметров**

Как вы видели в предыдущих разделах, значения из форм могут быть на верхнем уровне хэша params или вложены в другой хэш. Например, в стандартном экшне create для модели Person, params[:person] будет обычно хэшем всех атрибутов для создания персоны. Хэш params может также содержать массивы, массивы хэшей и тому подобное.

Фундаментально формы HTML не знают о структурировании данных, все, что они создают - это пары имя-значение, где пары являются обычными строками. Массивы и хэши, которые вы видите в своем приложении, - это результат некоторых соглашений по именованию параметров, которые использует Rails.

Вы можете быстрее пробовать примеры этого раздела, используя консоль для прямого вызова парсера параметров из Rack. Например:

|  |
| --- |
| Rack::Utils.parse\_query "name=fred&phone=0123456789"  # => {"name"=>"fred", "phone"=>"0123456789"} |

**Базовые структуры**

Две базовые структуры - это массивы и хэши. Хэши отражают синтаксис, используемый для доступа к значению в params. Например, если форма содержит

|  |
| --- |
| <input id="person\_name" name="person[name]" type="text" value="Henry"/> |

хэш params будет содержать

|  |
| --- |
| {'person' => {'name' => 'Henry'}} |

и params[:person][:name] получит отправленное значение в контроллере.

Хэши могут быть вложены на столько уровней, сколько требуется, например

|  |
| --- |
| <input id="person\_address\_city" name="person[address][city]" type="text" value="New York"/> |

вернет такой хэш params

|  |
| --- |
| {'person' => {'address' => {'city' => 'New York'}}} |

Обычно Rails игнорирует дублирующиеся имена параметра. Если имя параметра содержит пустой набор квадратных скобок [], то они будут накоплены в массиве. Если хотите, чтобы люди могли оставлять несколько телефонных номеров, можете поместить это в форму:

|  |
| --- |
| <input name="person[phone\_number][]" type="text"/>  <input name="person[phone\_number][]" type="text"/>  <input name="person[phone\_number][]" type="text"/> |

Что приведет к тому, что params[:person][:phone\_number] будет массивом.

**Комбинируем их**

Можно смешивать и сочетать эти две концепции. Например, один из элементов хэша может быть массивом, как в предыдущем примере, или вы можете иметь массив хэшей. Например, форма может позволить вам создать любое количество адресов, повторяя следующий фрагмент кода

|  |
| --- |
| <input name="addresses[][line1]" type="text"/>  <input name="addresses[][line2]" type="text"/>  <input name="addresses[][city]" type="text"/> |

Что приведет к тому, что params[:addresses] будет массивом хэшей с ключами line1, line2 и city. Rails начинает собирать значения в новый хэш, когда он встречает имя элемента, уже существующее в текущем хэше.

Однако, имеется ограничение, по которому хэши не могут быть вложены произвольно, является допустимым только один уровень "массивности". Массивы обычно могут быть заменены хэшами, например, вместо массива объектов моделей можно иметь хэш объектов модели с ключами, равными их id, индексу массива или любому другому параметру.

Параметры в массиве не очень хорошо работают с хелпером check\_box. В соответствии со спецификацией HTML, ненажатые чекбоксы не возвращают значения. Хелпер check\_box обходит это, создавая второе скрытое поле с тем же именем. Если чекбокс не нажат, подтверждается только скрытое поле, а если он нажат, то они оба подтверждаются, но значение от чекбокса получает приоритет. При работе с параметрами массива эти дублирующиеся подтверждения запутают Rails дублирующимися именами полей, и непонятно, как он решит, где начать новый элемент массива. Предпочтительнее использовать или check\_box\_tag, или хэши вместо массивов.

**Использование хелперов форм**

Предыдущие разделы совсем не использовали хелперы Rails. Хотя можно создавать имена полей самому и передавать их напрямую хелперам, таким как text\_field\_tag, Rails также предоставляет поддержку на более высоком уровне. В вашем распоряжении имеется два инструмента: параметр имени для form\_for и fields\_for, и опция :index, принимаемая этими хелперами.

Вы возможно захотите рендерить форму с набором полей ввода для каждого адреса человека. Например:

|  |
| --- |
| <%= form\_for @person do |person\_form| %>    <%= person\_form.text\_field :name %>    <% @person.addresses.each do |address| %>      <%= person\_form.fields\_for address, index: address do |address\_form|%>        <%= address\_form.text\_field :city %>      <% end %>    <% end %>  <% end %> |

Предположим, у кого-то есть два адреса с id 23 и 45, это создаст что-то подобное:

|  |
| --- |
| <form accept-charset="UTF-8" action="/people/1" class="edit\_person" id="edit\_person\_1" method="post">    <input id="person\_name" name="person[name]" type="text" />    <input id="person\_address\_23\_city" name="person[address][23][city]" type="text" />    <input id="person\_address\_45\_city" name="person[address][45][city]" type="text" />  </form> |

Это приведет к тому, что хэш params будет выглядеть так

|  |
| --- |
| {'person' => {'name' => 'Bob', 'address' => {'23' => {'city' => 'Paris'}, '45' => {'city' => 'London'}}}} |

Rails знает, что все эти поля должны быть частью хэша person, так как вы вызвали fields\_for для первого form builder. Определяя опцию :index, Вы сообщаете Rails, что вместо именования полей person[address][city], он должен вставить индекс, заключенный в [], между address и city. Если передать объект Active Record, как мы сделали, то Rails вызовет to\_param для него, который по умолчанию возвращает id в базе данных. Это часто полезно, так как просто обнаружить, какая запись Address должна быть изменена. Можете передать числа с некоторыми другими значениями, строки или даже nil (который приведет к созданию параметра в массиве).

Чтобы создать более замысловатые вложения, можете явно определить первую часть имени поля (person[address] в предыдущем примере), например

|  |
| --- |
| <%= fields\_for 'person[address][primary]', address, index: address do |address\_form| %>    <%= address\_form.text\_field :city %>  <% end %> |

создаст такие поля

|  |
| --- |
| <input id="person\_address\_primary\_1\_city" name="person[address][primary][1][city]" type="text" value="bologna" /> |

По общему правилу конечное имя поля - это сцепление имени, переданного в fields\_for/form\_for, значение индекса и имени атрибута. Можете также передать опцию :index прямо в хелперы, такие как text\_field, но, как правило, будет меньше повторов, если определить это на уровне form builder, чем для отдельного элемента ввода.

Как ярлык вы можете добавить [] к имени и опустить опцию :index. Это то же самое, что определение index: address, таким образом

|  |
| --- |
| <%= fields\_for 'person[address][primary][]', address do |address\_form| %>    <%= address\_form.text\_field :city %>  <% end %> |

создаст абсолютно тот же результат, что и предыдущий пример.

**Формы к внешним ресурсам**

Если необходимо передать через post некоторые данные внешнему ресурсу, было бы здорово создать форму, используя хелперы форм rails. Но иногда этому ресурсу необходимо передать authenticity\_token. Это можно осуществить, передав параметр authenticity\_token: 'your\_external\_token' в опциях form\_tag:

|  |
| --- |
| <%= form\_tag '<http://farfar.away/form>', authenticity\_token: 'external\_token') do %>    Form contents  <% end %> |

Иногда при отправке данных внешнему ресурсу, такому как платежный шлюз, поля, которые можно использовать в форме, ограничены внешним API. В этом случае вам может захотеться вообще не создавать скрытое поле authenticity\_token. Для этого нужно всего лишь передать false в опцию :authenticity\_token:

|  |
| --- |
| <%= form\_tag '<http://farfar.away/form>', authenticity\_token: false) do %>    Form contents  <% end %> |

Та же техника также доступна и для form\_for:

|  |
| --- |
| <%= form\_for @invoice, url: external\_url, authenticity\_token: 'external\_token' do |f| %>    Form contents  <% end %> |

Или, если не хотите создавать поле authenticity\_token:

|  |
| --- |
| <%= form\_for @invoice, url: external\_url, authenticity\_token: false do |f| %>    Form contents  <% end %> |

**Создание сложных форм**

Многие приложения вырастают из пределов простых форм, редактирующих одиночные объекты. Например, при создании Person вы, возможно, захотите позволить пользователю (в той же самой форме) создать несколько записей адресов (домашний, рабочий и т.д.). Позже, редактируя этого человека, пользователю должно быть доступно добавление, удаление или правка адреса, если необходимо.

**Настройка модели**

Active Record представляет поддержку на уровне модели с помощью метода accepts\_nested\_attributes\_for:

|  |
| --- |
| class Person < ActiveRecord::Base    has\_many :addresses    accepts\_nested\_attributes\_for :addresses    end    class Address < ActiveRecord::Base    belongs\_to :person  end |

Это создат метод addresses\_attributes= в Person, позволяющий создавать, обновлять и (опционально) уничтожать адреса.

**Создание формы**

Следующая форма позволяет пользователю создать Person и связанные с ним адреса.

|  |
| --- |
| <%= form\_for @person do |f| %>    Addresses:    <ul>      <%= f.fields\_for :addresses do |addresses\_form| %>        <li>          <%= addresses\_form.label :kind %>          <%= addresses\_form.text\_field :kind %>            <%= addresses\_form.label :street %>          <%= addresses\_form.text\_field :street %>          ...        </li>      <% end %>    </ul>  <% end %> |

Когда связь принимает вложенные атрибуты, fields\_for рендерит свой блок для каждого элемента связи. В частности, если у person нет адресов, он ничего не рендерит. Обычным паттерном для контроллера является построение одного или более пустых дочерних элементов, чтобы как минимум один набор полей был показан пользователю. Следующий пример покажет 3 набора полей адресов в форме нового person.

|  |
| --- |
| def new    @person = Person.new    3.times { @person.addresses.build}  end |

fields\_for вкладывает form builder, именующий параметры в формате, ожидаемом акцессором, созданным с помощью accepts\_nested\_attributes\_for. К примеру, при создании пользователя с 2 адресами, отправленные параметры будут выглядеть так

|  |
| --- |
| {    'person' => {      'name' => 'John Doe',      'addresses\_attributes' => {        '0' => {          'kind' => 'Home',          'street' => '221b Baker Street'        },        '1' => {          'kind' => 'Office',          'street' => '31 Spooner Street'        }      }    }  } |

Ключи хэша :addresses\_attributes не важны, они всего лишь должны быть различными для каждого адреса.

Если связанный объект уже сохранен, fields\_for автоматически создает скрытое поле с id сохраненной записи. Это можно отключить, передав include\_id: false в fields\_for. Это может быть желаемым, если автоматически созданное поле размещается туда, где тег input не имеет валидного HTML, или при использовании ORM, когда дочерние элементы не имеют id.

**Контроллер**

Как обычно, в контроллере необходим белый список параметров, который вы передаёте в модель:

|  |
| --- |
| def create    @person = Person.new(person\_params)    # ...  end    private    def person\_params      params.require(:person).permit(:name, addresses\_attributes: [:id, :kind, :street])    end |

**Удаление объектов**

Можно позволить пользователям удалять связанные объекты, передав allow\_destroy: true в accepts\_nested\_attributes\_for

|  |
| --- |
| class Person < ActiveRecord::Base    has\_many :addresses    accepts\_nested\_attributes\_for :addresses, allow\_destroy: true  end |

Если хэш атрибутов для объекта содержит ключ \_destroy со значением '1' или 'true', тогда объект будет уничтожен. Эта форма позволяет пользователям удалять адреса:

|  |
| --- |
| <%= form\_for @person do |f| %>    Addresses:    <ul>      <%= f.fields\_for :addresses do |addresses\_form| %>        <li>          <%= check\_box :\_destroy %>          <%= addresses\_form.label :kind %>          <%= addresses\_form.text\_field :kind %>          ...        </li>      <% end %>    </ul>  <% end %> |

Не забудьте обновить белый список параметров в вашем контроллере, а также включить туда поле \_destroy:

|  |
| --- |
| def person\_params    params.require(:person).      permit(:name, addresses\_attributes: [:id, :kind, :street, :\_destroy])  end |

**Предотвращение пустых записей**

Часто полезно игнорировать наборы полей, которые пользователь не заполнял. Этим можно управлять, передав :reject\_if proc в accepts\_nested\_attributes\_for. Этот proc будет вызван для каждого хэша атрибутов, отправляемого формой. Если proc возвращает false, тогда Active Record не создаст связанный объеккт для этого хэша. Следующий пример пытается создать адрес, если установлен атрибут kind.

|  |
| --- |
| class Person < ActiveRecord::Base    has\_many :addresses    accepts\_nested\_attributes\_for :addresses, reject\_if: lambda {|attributes| attributes['kind'].blank?}  end |

Вместо этого для удобства можно передать символ :all\_blank, который создаст proc, который отвергнет записи, когда все атрибуты пустые, за исключением любого значения для \_destroy.

**Добавление полей на лету**

Вместо рендеринга нескольких наборов полей, можно сделать их добавление только когда пользователь нажимает на кнопку 'Добавить новый элемент'. Rails не предоставляет какой-либо встроенной поддержки для этого. При создании новых наборов полей, следует убедиться, что ключ связанного массива уникальный - наиболее распространенным выбором является текущий javascript date (миллисекунды после epoch).