

Поддержка на стороне RubyOnRails

Метаинформация в моделях и ее публикация

Если серверная часть реализована на RoR, то метаинформацию удобнее всего располагать в классах моделей, ответственных за нужные сущности.

Для того, чтобы сделать это, следует описать метод `self.entity_meta`, который возвращает хеш с метаинформацией:

```
def self.entity_meta
  {
    opts: {...},
    cols:[...],
    includes: [...],
    ...
  }
end
```

Вторым шагом - добавить в ApplicationController метод `entities_init`, который заполняет хеш по используемым в EntUI сущностям.

Например:

```
def entities_init
  @entities = {
    "enum_type" => EnumType,
    "enum" => Enum,
    "doc_type" => DocType,
    "doc" => Doc
  }
end
```

Ключем хеша является название сущности, а значением - класс модели, которая эту сущность реализует.

И включаем этот метод в фильтр:

```
class ApplicationController < ActionController::Base
  before_filter :entities_init
end
```

Третий шаг, опубликовать метаинформацию в javascript - части.

Например, добавив во `views/layouts/application.html.erb` следующие строки:

```
<% if @entities %>
<script><!--
<% @entities.each do |k,v| %>
  EntUI_add_entity('<%= k %>', <%= raw v.entity_meta.to_json %>);
<% end %>
//--></script>
<% end %>
```

Четвертым шагом подключаем универсальный DataController, который обрабатывает ajax-запросы от EntUI.

Универсальный DataController

DataController работает со всеми сущностями подключенными к EntUI. Связано это с тем, что в параметре `id` запроса идет имя сущности (а класс модели DataController получает из `@entities` заполненного в ApplicationController).

Методы реализуемые DataController:

table

Метод table предоставляет список сущностей для плагина на основе DataTables. К сожалению, для поддержки другого табличного плагина, придется делать другой метод. То есть тут пока существует привязка. Соответственно, часть параметров наследуются от интерфейса DataTables.

Параметры:

- **id** - имя сущности.
- **sEcho** - код запроса. Должен возвращаться в json-ответе.
- **parent_type** - заполнено, если таблица рисуется на даунлинке. В этом случае в параметре имя сущности верхнего уровня.
- **parent_id** - заполнено, если таблица рисуется на даунлинке. В этом случае в параметре id сущности верхнего уровня.
- **iDisplayLength** - количество записей, которые следует вернуть для показа. Указывается для постраничного режима отображения таблицы. Иначе отсутствует или 0.
- **iDisplayStart** - номер начальной записи в списке. Указывается для постраничного режима отображения таблицы.
- **iSortCol_0** - номер первого поля в сортировке. В данной реализации сортировка больше чем по одному полю не поддерживается. Может быть не указано.
- **sSortDir_0** - направление сортировки. Может быть не указано.
- **параметры с именами атрибутов** из секции filters - значения выбранных фильтров. Могут отсутствовать.
- **sSearch** - Подстрока для поиска. Может отсутствовать. Поиск ведется по строковому представлению всех хранимых в базе атрибутов. Вычисляемые поля не затрагиваются.

Ключи возвращаемого объекта:

- **sEcho** - код запроса.
- **data** - список данных. В данном случае по формату список списков, где данные во вложенных списках соответствуют строке таблицы и должны идти в том же порядке, что и колонки (включая невидимые).
- **iTotalDisplayRecords** и **iTotalRecords** - количество записей для показа и количество записей всего. В данной реализации всегда совпадают.

Секция **includes** в метаданных позволяет оптимизировать запрос, сразу подключая объекты связанные по belongs_to, которые потребуются для формирования атрибутов таблицы.

filters

Метод возвращает трехуровневый список строк для заполнения блока комбо-боксов в фильтрах над таблицей.

Первый уровень - список фильтров, в той последовательности, как описано в секции filters метаданных.

Второй уровень - варианты для комбобокса.

Третий уровень - ["название", "значение"] или ["название оно же значение"]

Списки фильтров берутся как все различные значения поля имеющиеся в базе по каждому из полей (SELECT DISTINCT), либо как результат виртуальной функции filters.

form

Выдача данных объекта для построения формы.

Параметры:

- **id** - имя сущности.
- **entity_id** - id сущности.

Возвращает хеш с атрибутами описанными в секции form в метаданных, с некоторыми дополнениями:

- Всегда добавляется атрибут 'id', его не нужно описывать в секции form.
- Для всех атрибутов с kind=='link' добавляется дополнительный атрибут с именем "ИмяАтрибута_name". Тогда в атрибуте "ИмяАтрибута" идет id сущности на который указывает ссылка, а в атрибуте "ИмяАтрибута_name" - название/подпись на ссылке.

update

Создание новой сущности или сохранение изменений.

Параметры:

- **id** - имя сущности.
- **data** - атрибуты сущности.

Внутри параметра data может содержаться атрибут "id". Если он есть, то это сохранение измененной сущности, если нет - создание новой.

Возвращаемый объект, содержит результирующие значения атрибутов сущности после сохранения, среди них обязательно присутствует **"id"** (тот же или вновь полученный). Кроме того может присутствовать атрибут **"errors"** с текстом ошибки, если произошла ошибка записи или валидации.

Виртуальные методы в моделях

Виртуальные методы позволяют кастомизировать работу универсального DataController.

При стандартной обработке никакие из них не нужны.

self.custom_parent(req, parent_type, parent_id)

Обработка запроса на даунлинки.

- **req** - частично заполненный запрос (объект класса ActiveRecord::QueryMethods)
- **parent_type** - имя родительской сущности (строка)
- **parent_id** - id родительской записи.

Метод должен вернуть или измененный req или nil. В первом случае стандартная обработка parent_id в контроллере отключается (считается, что все нужные добавки в запрос уже внесены).

self.custom_order(req, sort_atr, sort_dir)

Обработка сортировки в запросе таблицы.

- **req** - частично заполненный запрос (объект класса ActiveRecord::QueryMethods).
- **sort_atr** - имя атрибута в таблице.
- **sort_dir** - asc или desc - направление сортировки.

Метод должен вернуть или измененный req или nil. В первом случае стандартная обработка в контроллере отключается.

self.custom_filters(req, atr, vaue)

Фильтрация на основе блока комбо-боксов.

- **req** - частично заполненный запрос (объект класса ActiveRecord::QueryMethods).
- **atr** - имя атрибута в таблице.
- **vaue** - значение атрибута.

Метод вызывается для каждого пришедшего фильтра.

Метод должен вернуть или измененный req или nil. В первом случае стандартная обработка в контроллере отключается.

self.filters(atr)

Значения для комбо-боксов блока фильтров.

- **req** - частично заполненный запрос (объект класса ActiveRecord::QueryMethods).
- **atr** - имя атрибута в таблице.

Возвращает или двухуровневый список тиа [{"название", "значение"}, {"название оно же значение"}, ...] или nil.

Если возвращается nil, используется стандартный алгоритм обработки.

Вычисляемые поля

DataController обращается к данным модели не через атрибуты, а через методы. Поэтому, с точки зрения получения данных (например, в запросе form) нет никакой разницы, получено значение атрибута из базы или вычислено в классе модели.

(, , .)

Поэтому, напрмер можно описать такой атрибут:

```
cols:[
  {atr: 'external_mark', label: 'Внешний?'},
  ...
]
```

Поддерживаемый методом модели:

```
def external_mark
  self.external ? '*' : "
end
```

Сложности с атрибутами для аплинков

Для описания атрибутов апликов возможно несколько подходов, но пока все оне содержат неудобства.

Для пример рассмотрим ситуацию, когда у нас есть сущность "сообщение" (message) и родительская сущность "тема"(subject).

По стандартным для RoR соглашениям, в классе Message присутсвует атрибут subject_id и связанный объект subject.

```
class Message < ActiveRecord::Base
  belongs_to :subject
```

При этом нам нужно наличие трех методов:
atrName(), atrName_name(), atrName=(value)

Приведу два способа описать эту ситуацию для EntUI.

Короткий способ

Называем атрибут "subject_id". Тогда получается, что методы subject_id() и subject_id=(value) у нас есть и надо создать метод subject_id_name().

```
def subject_id_name()
  self.subject.name
end
```

Предположим, что название темы мы хотим отобразить в таблице сообщений:

```
class Message < ActiveRecord::Base
  belongs_to :subject

  def self.entity_meta
    {
      cols:[
        {atr: 'id', label: '#'},
        {atr: 'subject_id_name', label: 'Тема'},
        ...
      ],
      includes: ['subject'],
    }
  end

  def subject_id_name()
    self.subject.title
  end

  ...
end
```

Поскольку метод будет вызываться для каждой строки, есть смысл добавить используемый им объект в includes. Но внимание! Туда вписывается имя, которое стоит под belongs_to.

Дальше подключаем этот аплинк к форме:

```
class Message < ActiveRecord::Base
  belongs_to :subject

  def self.entity_meta
    {
      cols:[
        {atr: 'id', label: '#'},
        {atr: 'subject_id_name', label: 'Тема'},
        ...
      ],
      includes: ['subject'],
      form:[
        {atr: 'subject_id', label:'Тема', kind:'link', entity: 'subject', name_atr: 'title'}
        ...
      ],
    }
  end

  def subject_id_name()
    self.subject.title
  end

  ...
end
```

Если бы текстовое поле в subject называлось name, опцию name_atr можно было бы не указывать. А title не является

значением по умолчанию.

Все! Не очень много, но имена атрибутов получаются не красивыми.

Длинный способ

Но предположим, что мы хотим, чтобы атрибуты назывались красиво: `subject` и `subject_name`.

Тогда первое, что придется сделать - освободить имя под `belongs_to`, а второе - описать все три метода.

```
class Message < ActiveRecord::Base
  belongs_to :parent, class_name: 'Subject', foreign_key: 'subject_id'

  def self.entity_meta
    {
      cols:[
        {atr: 'id', label: '#'},
        {atr: 'subject_name', label: 'Тема'},
        ...
      ],
      includes: ['parent'],
      form:[
        {atr: 'subject', label:'Тема', kind:'link', entity: 'subject', name_atr: 'title'}
        ...
      ],
    }
  end

  def subject_name()
    self.parent.title
  end

  def subject()
    self.subject_id
  end

  def subject= (value)
    self.subject_id = value
  end

  ...
end
```

Обратите внимание: под `includes` сейчас стоит `'parent'`

Используемые библиотеки

```
gem 'rails', '~> 4.2'
gem 'squeel', '~> 1.2.3'
```