**Тема занятия № 27: Модуль 14. Наборы форм, связанные с моделями**

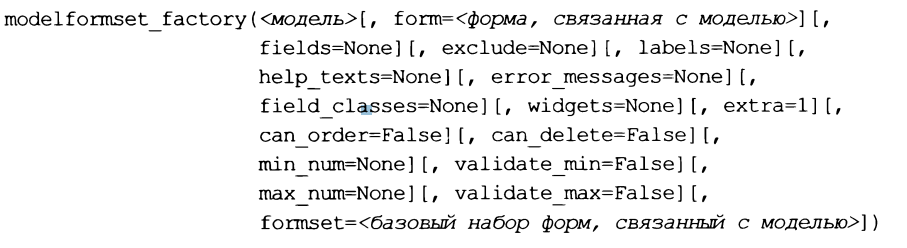
1. Создание наборов форм, связанных с моделями

Если обычная форма, связанная с моделью, позволяет работать лишь с одной записью, то набор форм, связанный с моделью, дает возможность работы сразу с несколькими записями. Внешне он представляет собой группу форм, в каждой из которых отображается содержимое одной записи. Помимо того, там могут быть выведены “пустые” формы для добавления записей и специальные средства для переупорядочивания и удаления записей.

Можно сказать, что один набор форм, связанный с моделью, заменяет несколько страниц: страницу списка записей и страницы добавления, правки и удаления записей. Вот только, к сожалению, с наборами форм удобно работать лишь тогда, когда количество отображающихся в них записей невелико.

**Создание наборов форм, связанных с моделями**

Для создания наборов форм, связанных с моделями, применяется быстрое объявление посредством фабрики классов— функции modeiformset\_factory() из модуля.

****

Параметров здесь очень много:

□ первый, позиционный — модель, связываемая с формируемым набором форм;

О form— форма, связанная с моделью, на основе которой будет создан набор форм. Если параметр не указан, то форма будет создана автоматически;

□ fields — последовательность имен полей модели, включаемых в форму, автоматически создаваемую для набора. Чтобы указать все поля модели, нужно присвоить этому параметру строку "\_\_ail\_\_";

□ exclude — последовательность имен полей модели, которые, напротив, не должны включаться в форму, автоматически создаваемую для набора;

Внимание!

В вызове функции modelformset\_factory() должвн присутствовэть ТОЛЬКО ОДИН ИЗ слвдующих параметров: form, fields, exclude. Одновременное указание двух или более параметров приведет к ошибке.

□ labels — надписи для полей формы. Указываются в виде словаря, ключи элементов которого соответствуют полям, а значения задают надписи для них;

□ help texts — дополнительные текстовые пояснения для полей формы. Указывается в виде словаря, ключи элементов которого соответствуют полям, а значения задают пояснения для них;

□ error messages — сообщения об ошибках. Задаются в виде словаря, ключи элементов которого соответствуют полям формы, а значениями элементов также должны быть словари. Во вложенных словарях ключи элементов соответствуют строковым кодам ошибок, а значения зададут строковые сообщения об ошибках;

□ fieid ciasses — типы полей формы, которыми будут представляться в создаваемой форме различные поля модели. Значением должен быть словарь, ключи элементов которого совпадают с именами полей модели, а значениями элементов станут ссылки на классы полей формы;

□ widgets — элементы управления, представляющие различные поля формы. Значение — словарь, ключи элементов которого совпадают с именами полей формы, а значениями элементов станут экземпляры классов элементов управления или ссылки на сами эти классы;

Внимание!

Параметры labels, help\_texts, error\_messages, field\_classes И widgets указываются только в том случае, если форма для набора создается автоматически (параметр form не указан). В противном случае все необходимые сведения о форме должны быть записаны в ее классе.

□ extra — количество ’’пустых” форм, предназначенных для ввода новых записей, которые будут присутствовать в наборе (по умолчанию — 1);

□ can order — если True, то посредством набора форм можно переупорядочивать записи связанной с ним модели, если False — нельзя (поведение по умолчанию);

□ can delete — если True, то посредством набора форм можно удалять записи связанной с ним модели, если False — нельзя (поведение по умолчанию);

□ min num— минимальное количество форм в наборе, за вычетом помеченных на удаление (по умолчанию не ограничено);

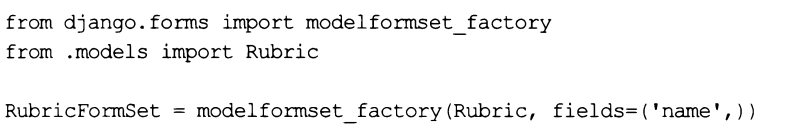
□ vaiidate min— если True, то Django в процессе валидации будет проверять, не меньше ли количество форм значения из параметра min num, если False — не будет (поведение по умолчанию). Если количество форм меньше указанного минимального, будет выведено сообщение об ошибке с кодом "toofewforms" (этот код можно использовать для указания своего сообщения об ошибке);

□ max num— максимальное количество форм в наборе, за вычетом помеченных на удаление (по умолчанию не ограничено);

□ vaiidate max — если True, то Django в процессе валидации будет проверять, не превосходит ли количество форм значение из параметра max num, если False — не будет (поведение по умолчанию). Если количество форм больше указанного максимального, будет выведено сообщение об ошибке с кодом "too many forms";

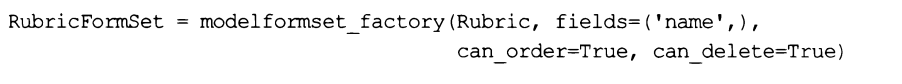
□ formset — базовый набор форм, связанный с моделью, на основе которого будет создан новый набор форм. Он должен быть производным от класса Basemodelformset из модуля django. Forms и может указывать какие-либо общие для целой группы наборов форм функции.

Пример создания набора форм, предназначенного для работы со списком рубрик и имеющего минимальную функциональность:



На экране он будет выглядеть как простая последовательность форм, каждая из которых служит для правки одной записи модели (отображение на внешнем виде).

А вот пример создания аналогичного набора форм, позволяющего переупорядочивать и удалять рубрики:

****

Набор форм с подобного рода расширенной функциональностью показан на выше. Видно, что в составе каждой формы находятся поле ввода Порядок и флажок Удалить. В поле ввода заносится целочисленное значение, по которому записи модели могут быть отсортированы. А установка флажка приведет к тому, что после нажатия на кнопку отправки данных соответствующие записи будут удалены из модели.

1. Обработка наборов форм, связанных с моделями

Высокоуровневые контроллеры-классы, не "умеют" работать с наборами форм. Поэтому их в любом случае придется обрабатывать вручную.

**Создание набора форм, связанного с моделью**

Экземпляр набора форм создается вызовом конструктора его класса без параметров:

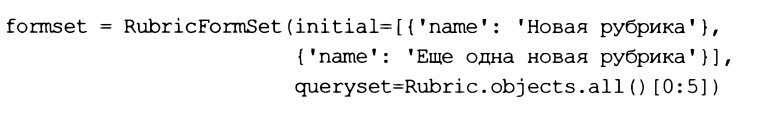


Конструктор класса набора форм поддерживает два необязательных параметра, которые могут пригодиться:

□ initial — изначальные данные, которые будут помещены в ’’пустые” (предназначенные для добавления новых записей) формы. Значение параметра должно представлять собой последовательность, каждый элемент которой задаст изначальные значения для одной из ’’пустых” форм. Этим элементом должен выступать словарь, ключи элементов которого совпадают с именами полей ’’пустой” формы, а значения элементов зададут изначальные значения для этих полей;

□ queryset — набор записей для вывода в наборе форм.

Для примера зададим в качестве изначального значения для поля названия в первой ’’Пустой” форме строку "Новая рубрика", ВО второй — Строку "Еще одна новая рубрика" и сделаем так, чтобы в наборе форм выводились только первые пять рубрик:

****

**Повторное создание набора форм**

Закончив работу, посетитель нажмет кнопку отправки данных, в результате чего веб-обозреватель отправит POST-запрос с данными, занесенными в набор форм.

Этот запрос нужно обработать. Прежде всего, следует создать набор форм повторно. Это выполняется так же, как и в случае формы, — вызовом конструктора класса с передачей ему единственного позиционного параметра — словаря из атрибута post объекта запроса. Пример:

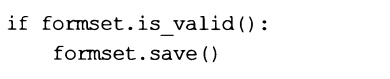


Если при первом создании набора форм в параметре queryset был указан набор записей, то при повторном создании также следует его указать:



**Валидация и сохранение набора форм**

Валидация и сохранение набора форм выполняется методами is valid (), save () и save\_m2m (), поддерживаемыми классом набора форм:

****

Метод save () в качестве результата вернет последовательность всех записей модели, представленных в текущем наборе форм. Эту последовательность можно перебрать в цикле и выполнить над записями какие-либо действия (что может пригодиться при вызове метода save () С параметром commit, равным False).

После вызова метода save () будут доступны три атрибута, поддерживаемые классом набора форм:

О new objects — последовательность добавленных записей модели, связанной с текущим набором форм;

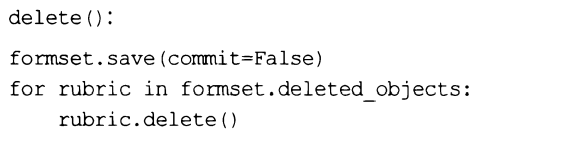
□ changed\_objects— последовательность исправленных записей модели, связанной с текущим набором форм;

□ deleted\_objects— последовательность удаленных записей модели, связанной с текущим набором форм.

Метод save о самостоятельно обрабатывает удаление записей (если при вызове конструктора набора форм был указан параметр can delete со значением True).

Если посетитель в форме установит флажок Удалить, то соответствующая запись модели будет удалена.

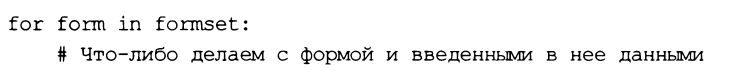
Однако при вызове метода save о с параметром commit, равным False, помеченные на удаление записи удалены не будут. Понадобится перебрать все удаленные записи, перечисленные в списке из атрибута deieted\_objects, и вызвать у каждой метод Delete ():

****

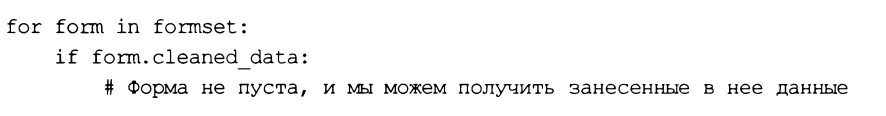
**Доступ к данным, занесенным в набор форм**

Каждая форма, входящая в состав набора, поддерживает атрибут cieaned data. Его значением является словарь, хранящий все данные, которые были занесены в текущую форму, в виде объектов языка Python.

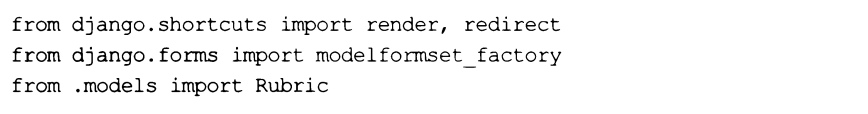
Сам набор форм поддерживает функциональность последовательности. На каждой итерации он возвращает очередную форму, входящую в его состав. Вот так можно перебрать в цикле входящие в набор формы:

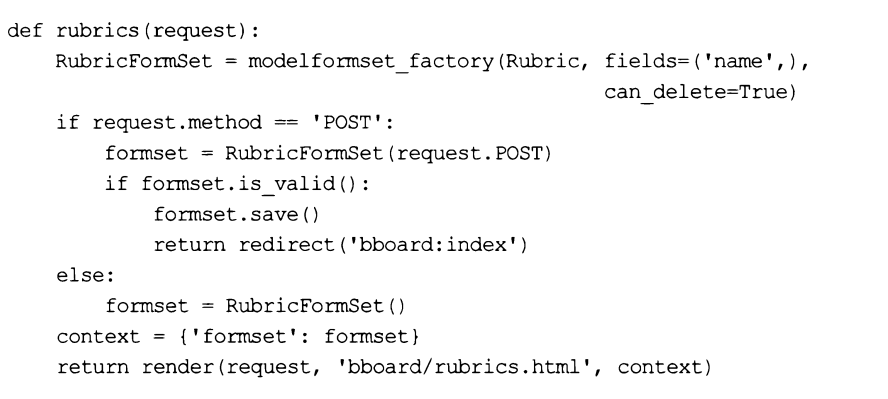


Нужно иметь в виду, что в наборе, возможно, будет присутствовать ’’пустая” форма, в которую не были занесены никакие данные. Такая форма будет хранить в атрибуте cieaned data ’’пустой” словарь. Обработать эту ситуацию можно следующим образом:



Полный код контроллера-функции, обрабатывающего набор форм и позволяющего удалять записи.





**Реализация переупорядочивания записей**

К сожалению, метод save () не выполняет переупорядочивание записей, и его придется реализовывать вручную.

Переупорядочивание записей достигается тем, что в каждой форме, составляющей набор, автоматически создается целочисленное поле с именем, заданным в переменной ORDERING\_FIELD\_NAME ИЗ модуля dj ango. Forms. Formsets. В ЭТОМ поле хранится целочисленный порядковый номер текущей записи в последовательности.

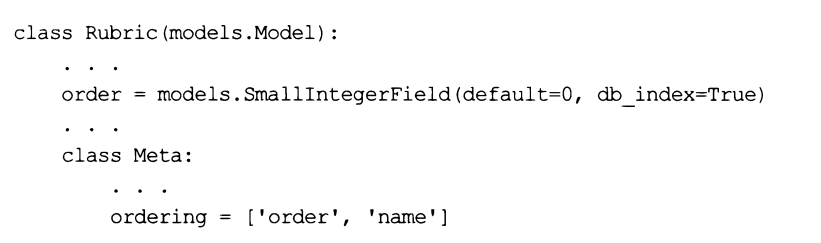
При выводе набора форм в такие поля будут подставлены порядковые номера записей, начиная с 1. Меняя эти номера, посетитель и переупорядочивает записи.

Чтобы сохранить заданный порядок записей, указанные для них значения порядковых номеров нужно куда-то записать. Для этого следует:

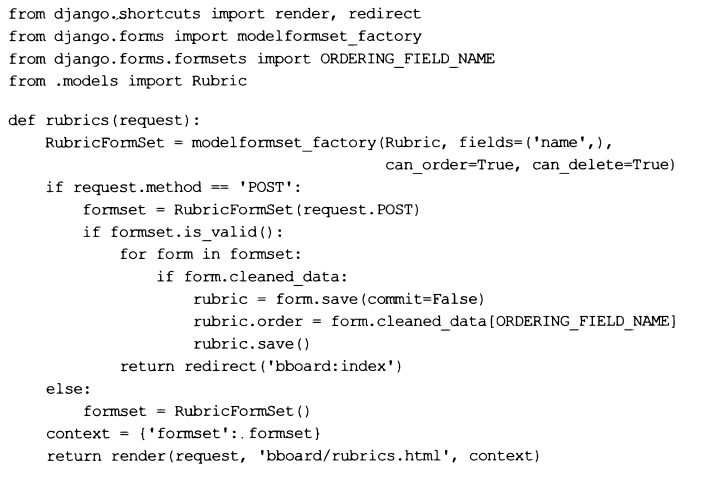
1. Добавить в модель поле целочисленного типа.

2. Указать порядок сортировки по значению этого поля (обычно по возрастанию — так будет логичнее).

Например, для переупорядочивания рубрик в модели Rubric можно предусмотреть поле order\*.



Полный код контроллера-функции, обрабатывающего набор форм, позволяющего удалять и переупорядочивать записи. Обратим внимание, как порядковые номера записей сохраняются в поле order модели Rubric.



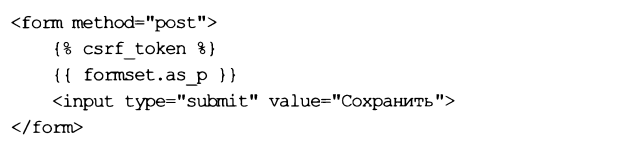
1. Вывод наборов форм на экран

В целом вывод наборов форм на экран выполняется так же и теми же средствами, что и вывод обычных форм.

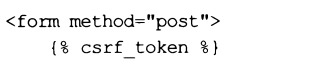
**Быстрый вывод наборов форм**

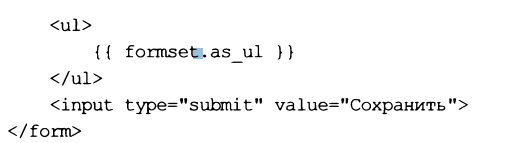
Быстрый вывод наборов форм выполняют три следующих метода:

□ as\_p() — вывод по абзацам. Надпись для элемента управления и сам элемент управления каждой формы из набора выводятся в отдельном абзаце и отделяются друг от друга пробелами. Пример использования:



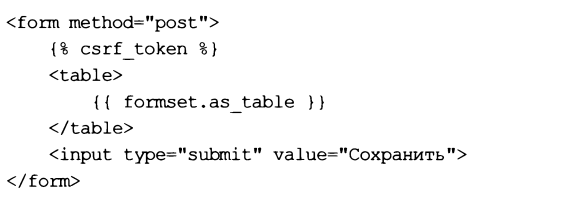
□ as ui () — вывод в виде маркированного списка. Надпись и сам элемент управления выводятся в отдельном пункте списка и отделяются друг от друга пробелами. Теги <ui> и </ui> не формируются. Пример:



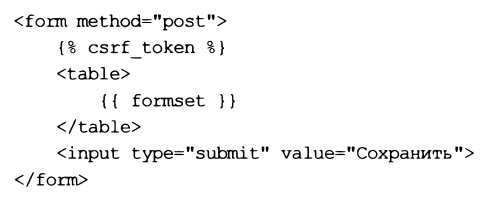


□ as tabie () — вывод в виде таблицы с двумя столбцами: в левом выводятся надписи, в правом — элементы управления. Каждая пара ’’надпись—элемент управления” занимает отдельную строку таблицы. Теги <tabie> и </tabie> не выводятся.

Пример:



Можно просто указать переменную, хранящую набор форм, — метод as tabie () будет вызван автоматически:

****

Парный тег <form>, создающий саму форму, в этом случае не создается, равно как и кнопка отправки данных. Также следует поместить в форму тег шаблонизатора Csrf token, который создаст скрытое поле с электронным жетоном безопасности.

**Расширенный вывод наборов форм**

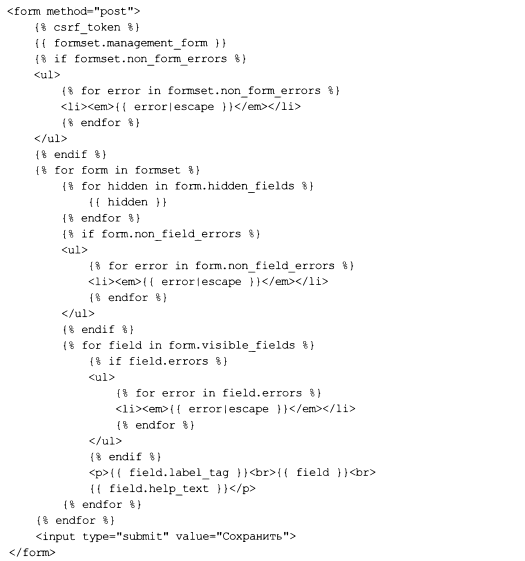
Инструментов для расширенного вывода наборов форм Django предоставляет совсем немного.

Прежде всего, это атрибут management form, поддерживаемый всеми классами наборов форм. Он хранит ссылку на служебную форму, входящую в состав набора и хранящую необходимые для работы служебные данные.

Далее, не забываем, что набор форм поддерживает функциональность итератора, возвращающего на каждом проходе очередную входящую в него форму.

И наконец, метод non form errors () набора форм возвращает список сообщений об ошибках, относящихся ко всему набору.

Код шаблона bboard\rubrics.html.



1. Валидация в наборах форм

Валидацию удобнее всего реализовать:

□ в модели, с которой связан набор форм;

□ в связанной с этой моделью форме, на основе которой создается набор. Эта форма задается в параметре form функции modelformset factory ().

В самом наборе форм имеет смысл выполнять валидацию лишь тогда, когда необходимо проверить весь массив данных, введенных в этот набор.

Валидация в наборе форм реализуется так:

1. Объявляется класс, производный от класса basemodeiformset из модуля Django.forms.

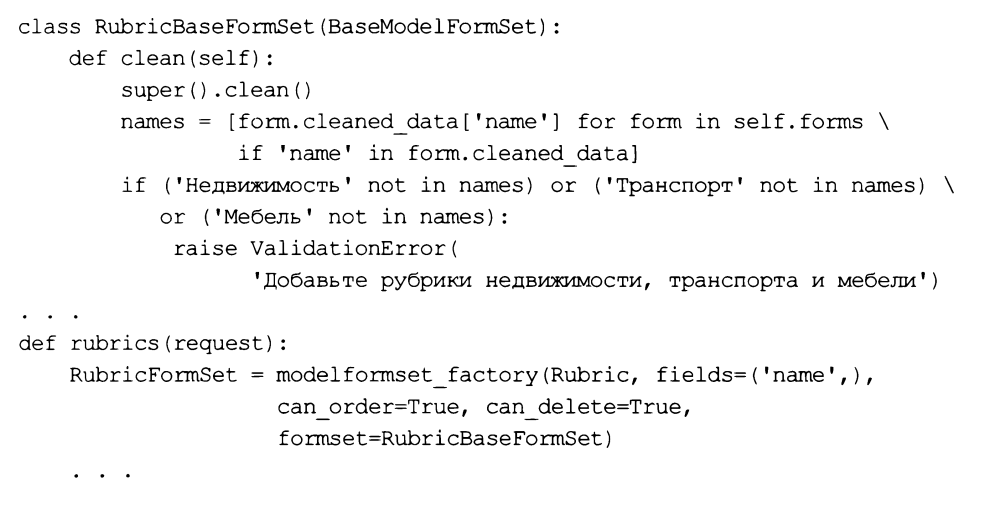
2. В этом классе переопределяется метод clean (self), в котором и выполняется валидация. Этот метод должен удовлетворять тем же требованиям, что и одноименный метод класса формы, связанной с моделью.

3. Создается набор форм — вызовом функции modelformset factory (), в параметре Formset которой указывается объявленный класс набора форм.

Атрибут forms, унаследованный от класса basemodelformset, хранит последовательность всех форм, что имеются в наборе.

Сообщения об ошибках, генерируемые таким валидатором, будут присутствовать в списке ошибок, относящихся ко всему набору форм (возвращается методом Non\_form\_errors()).

Вот пример кода, выполняющего валидацию на уровне набора форм и требующего обязательного присутствия рубрик ’’Недвижимость”, ’’Транспорт” и ’’Мебель”:

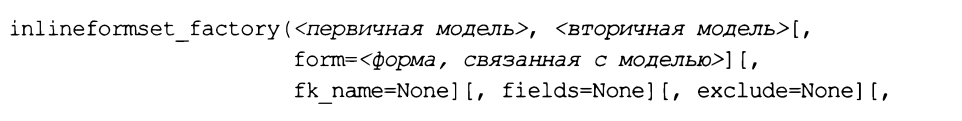


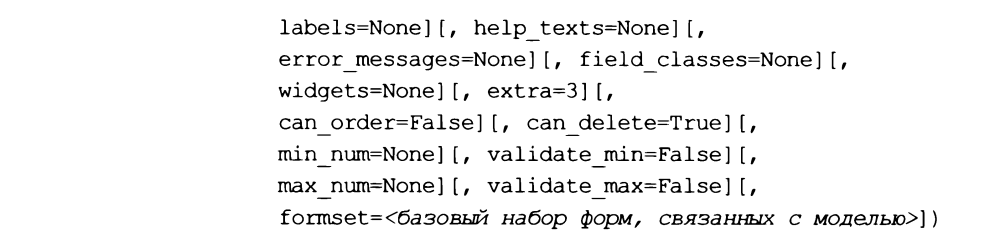
1. Встроенные наборы форм

Встроенные наборы форм служат для работы с записями вторичной модели, связанными с указанной записью первичной модели.

**Создание встроенных наборов форм**

Для создания встроенных наборов форм применяется функция inlineformset\_Factory () из модуля django. Forms. Вот формат ее вызова:





В первом позиционном параметре указывается ссылка на класс первичной модели, а во втором позиционном параметре — ссылка на класс вторичной модели.

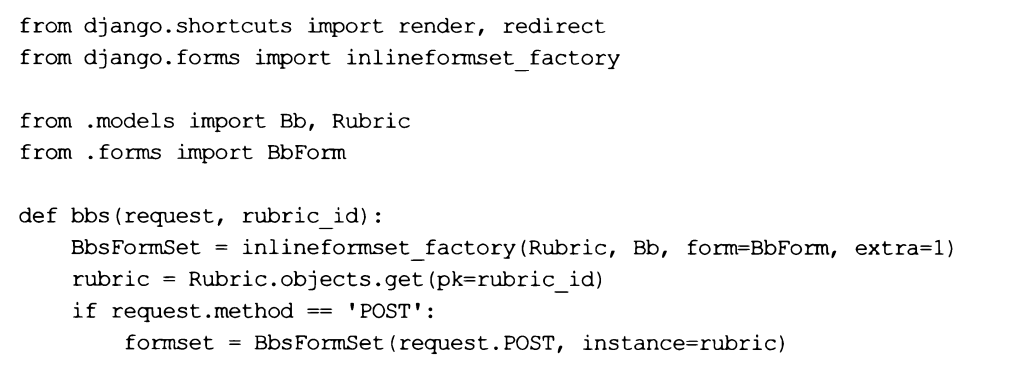
Параметр fk\_name задает имя поля внешнего ключа вторичной модели, по которому устанавливается связь с первичной моделью, в виде строки. Он указывается только в том случае, если во вторичной модели установлено более одной связи с первичной моделью и требуется выбрать, какую именно связь нужно использовать.

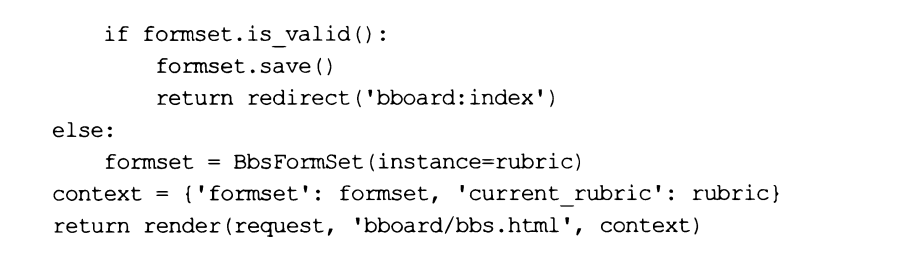
Остальные параметры имеют такое же назначение, что и у функции modelformsetfactory. Отметим только, что у параметра extra (задает количество "пустых" форм) значение по умолчанию 3, а у параметра can delete (указывает, можно ли удалять связанные записи) — True. А базовый набор форм, задаваемый в параметре formset, должен быть производным ОТ класса baselnlineformset ИЗ модуля django. Forms.

**Обработка встроенных наборов форм**

Обработка встроенных наборов форм выполняется так же, как и обычных наборов форм, связанных с моделями. Только при создании объекта набора форм как в первый, так и во второй раз нужно передать конструктору с параметром instance запись первичной модели. После этого набор форм выведет записи вторичной модели, связанные с ней.

Код контроллера, который выводит на экран страницу со встроенным набором форм, который показывает все объявления, относящиеся к выбранной посетителем рубрике, позволяет править и удалять их. В нем используется форма bbform.





Вывод встроенного набора форм на экран выполняется с применением тех же программных инструментов, что и вывод обычного набора форм, связанного с моделью.

И точно такими же средствами во встроенном наборе форм реализуется валидация.

Единственное исключение: объявляемый для этого класс должен быть производным ОТ класса baselnlineformset ИЗ МОДУЛЯ django. Forms.