**Тема занятия № 34: Административный сайт Django. Отправка электронных писем**

1. Подготовка административного веб-сайта к работе

Django включает в свой состав полностью готовый к работе административный веб-сайт, предоставляющий доступ к любым внутренним данным, позволяющий пополнять, править и удалять их, гибко настраиваемый и исключительно удобный в использовании.

Доступ к административному сайту Django имеют только суперпользователь и пользователи со статусом персонала.

Прежде чем работать с административным сайтом, необходимо осуществить следующие подготовительные операции:

□ открыть модуль настроек проекта settings.py, находящийся в пакете конфигурации, и проверить:

• присутствуют ли в списке зарегистрированных в проекте приложений (параметр INSTALLED\_APPS) приложения django. Contrib. Admin, django. Contrib. Auth, Django.contrib.contenttypes, django.contrib.messages И django.contrib.

Sessions:

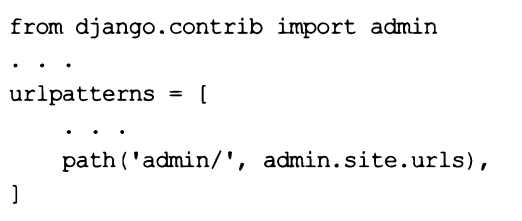
• присутствуют ли в списке зарегистрированных посредников (параметр MIDDLEWARE) посредники dj ango. Contrib. Auth .middleware. Authenticationmiddleware, django.contrib.sessions.middleware.sessionmiddleware И django.

Contrib.messages.middleware.messagemiddleware;

• проверить, присутствуют ли в списке зарегистрированных для используемого шаблонизатора обработчиков контекста (вложенный параметр context Processors параметра OPTIONS) обработчики django. Contrib. Auth. Context\_Processors. Auth И django. Contrib. Messages. Context\_processors .messages;

□ добавить в список маршрутов уровня проекта (модуль uris.py пакета конфигурации) маршрут, который свяжет выбранный шаблонный путь (обычно используется admin/) со списком маршрутов, записанным в атрибуте urls объекта административного сайта, который хранится в переменной site из модуля django.

Contrib. Admin:

****

Впрочем, во вновь созданном проекте все эти действия выполнены изначально;

□ выполнить миграции;

□ создать суперпользователя.

На заметку!

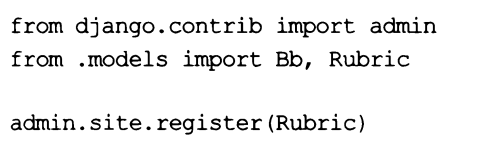
Для своих нужд Django создает в базе данных таблицу django admin iog, хранящую "историю" манипуляций с содержимым базы данных посредством административного сайта.

1. Регистрация моделей на административном веб-сайте

Чтобы с данными, хранящимися в определенной модели, можно было работать посредством административного сайта, модель нужно зарегистрировать на сайте.

Делается это вызовом метода register (<модель>) объекта административного сайта.

Пример:

****

Код, выполняющий регистрацию моделей на административном сайте, следует помещать в модуль admin.py пакета приложения.

Это самый простой способ сделать модель доступной через административный сайт. Однако в этом случае записи, хранящиеся в модели, будут иметь представление по умолчанию: отображаться в виде их строкового представления (которое формируется методом\_\_str\_\_()), выводиться в порядке их добавления, не поддерживать специфических средств поиска и пр.

Если нужно, чтобы список записей представлялся в виде таблицы с колонками, в которых выводятся значения их отдельных полей, а также требуется получить доступ к специфическим средствам поиска и настроить внешний вид страниц добавления и правки записей, то придется создать для этой модели класс редактора.

1. Редакторы моделей

Редактор модели — это класс, указывающий параметры представления модели на административном сайте: набор полей, выводящихся на экран, порядок сортировки записей, применение специальных средств для их фильтрации, элементы управления, используемые для занесения значений в поля записей, и пр.

Класс редактора должен быть производным от класса modeladmin из модуля Django.contrib.admin. Его объявление, равно как и регистрирующий его код, следует записывать в модуле admin.py пакета приложения. Пример объявления класса-редактора можно далее.

Различные параметры представления модели записываются в классе редактора в виде его атрибутов или методов.

**Параметры списка записей**

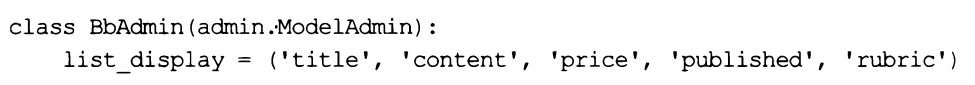
Страница списка записей выводит перечень записей, хранящихся в модели, позволяет осуществлять их фильтрацию, сортировку, выполнять различные действия над группой выбранных записей (в частности, их удаление) и даже, при указании соответствующих настроек, править записи, не заходя на страницу правки.

**Параметры списка записей: состав выводимого списка**

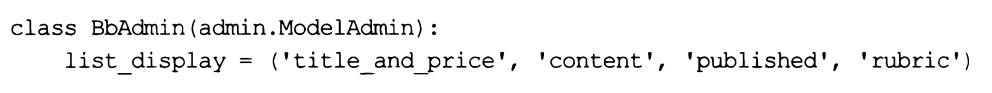
Параметры из этого раздела задают поля, выводимые в списке записей, и возможность правки записей непосредственно на странице списка:

□ list display— атрибут, задает набор выводимых в списке полей. Его значением должен быть список или кортеж, элементом которого может быть:

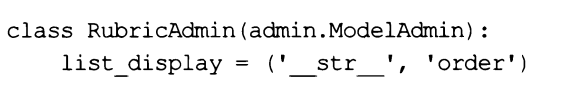
• имя поля модели в виде строки:

****

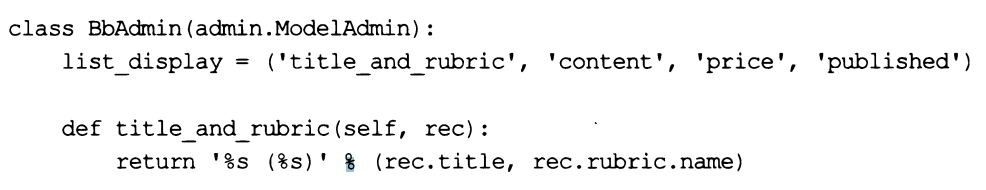
• имя функционального поля, объявленного в модели, в виде строки. Вот пример указания функционального ПОЛЯ title\_and\_price модели Bb:

****

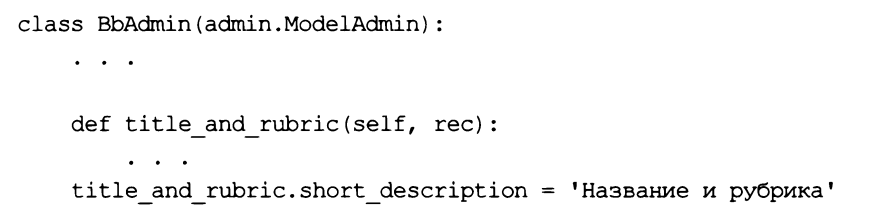
Также можно указать имя метода\_\_str\_\_о, который формирует строковое представление модели:

****

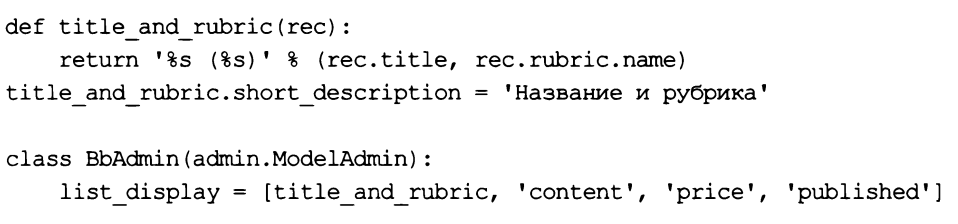
• имя функционального поля, объявленного непосредственно в классе редактора, также в виде строки. Такое поле реализуется методом, принимающим в качестве единственного параметра объект записи и возвращающим результат, который и будет выведен на экран. Вот пример указания функционального ПОЛЯ title\_and\_rubric, объявленного в классе редактора bbadmin:



У созданного таким образом функционального поля можно указать название, присвоив его атрибуту short description объекта метода, реализующего поле:

****

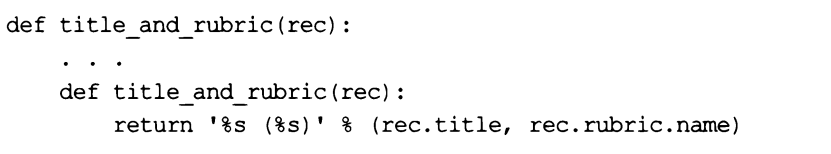
• ссылка на функциональное поле, объявленное в виде обычной функции. Такая функция должна принимать с единственным параметром объект записи и возвращать результат, который и будет выведен на экран. Для этого поля можно задать название, присвоив его атрибуту short description объекта функции. Вот пример указания функционального поля title and rubric, реализованного в виде обычной функции:

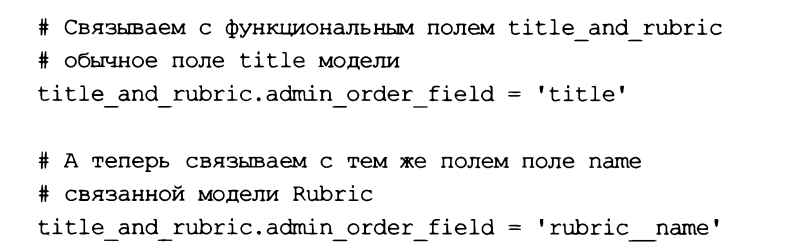


Поля типа manytomanyfieid не поддерживаются, и их содержимое не будет выводиться на экран.

По значениям функциональных полей невозможно выполнять сортировку, поскольку записи сортируются СУБД, которая не ’’знает” о существовании функциональных полей. Однако можно связать функциональное поле с обычным полем модели, и тогда щелчок на заголовке функционального поля на странице списка записей приведет к сортировке записей по значениям указанного обычного поля. Для этого достаточно присвоить строку с именем связываемого обычного поля модели атрибуту admin order field объекта метода (функции),

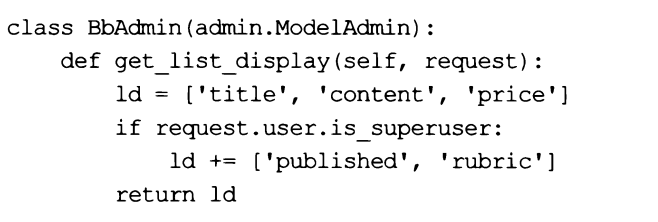
Реализующего функциональное поле. Вот пара примеров:





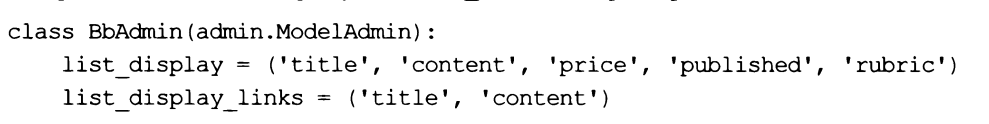
□ get\_list\_display(self, request) — метод, должен возвращять набор ВЫВОДИМЫХ в списке полей. Полученный с параметром request запрос можно использовать при формировании этого набора.

В данном примере обычным пользователям показываются только название, описание и цена товара, а суперпользователю — также рубрика и временная отметка публикации:



В изначальной реализации метод возвращает значение атрибута list display;

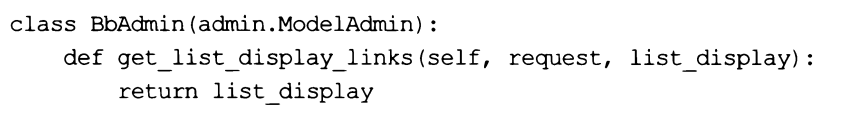
□ list display links — атрибут, задает перечень полей, значения которых будут превращены в гиперссылки, указывающие на страницы правки соответствующих записей. В качестве значения указывается список или кортеж, элементами которого являются строки с именами полей. Эти поля должны присутствовать в перечне, заданном атрибутом list display. Пример:



Если в качестве значения атрибута задать ’’пустой” список или кортеж, то в гиперссылку будет превращено значение самого первого поля, выводящегося в списке, а если указать None, то такие гиперссылки вообще не будут создаваться. Значение по умолчанию — ’’пустой” список;

О get\_list\_display\_links (self, request, list\_display) —метод, должен возвращать перечень полей, значения которых будут превращены в гиперссылки на страницы правки соответствующих записей. Параметром request передается запрос, а параметром list display — список с именами выводимых полей, который может быть использован для создания перечня полей-гиперссылок.

Вот пример преобразования в гиперссылки всех полей, выводимых на экран:



В изначальной реализации метод возвращает значение атрибута iist\_dispiay\_Links, если его значение не равно ’’пустому” списку или кортежу. В противном случае возвращается список, содержащий первое поле из состава выводимых на экран;

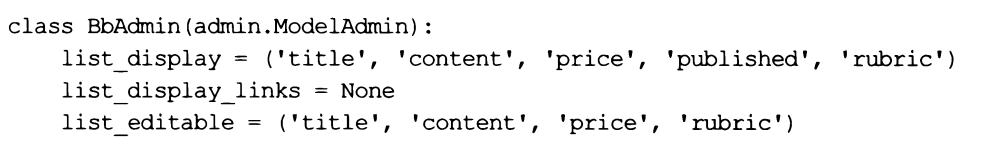
□ iist\_editable— атрибут, задает перечень полей, которые можно будет править непосредственно на странице списка записей, не переходя на страницу правки.

В соответствующих столбцах списка записей будут присутствовать элементы управления, с помощью которых пользователь сможет исправить значения полей, указанных в перечне. После правки необходимо нажать расположенную в нижней части страницы кнопку Сохранить, чтобы внесенные правки были сохранены.

В качестве значения атрибута указывается список или кортеж, элементами которого должны быть имена полей, представленные в виде строк. Если указать ’’Пустой” список или кортеж, то ни одно поле модели не может быть исправлено на странице списка.

Внимание!

Поля, указанные в перечне из атрибута Ust editabie, не должны присутствовать в перечне из атрибута iist\_dispiay\_iinks. Пример:

****

Значение по умолчанию — ’’пустой” кортеж;

□ Ust seiect reiated— атрибут, устанавливает набор первичных моделей, чьи связанные записи будут извлекаться одновременно с записями текущей модели посредством вызова метода select related о. В качестве значения можно указать:

• True —извлекать связанные записи всех первичных моделей, связанных с текущей моделью;

• False — извлекать связанные записи только тех первичных моделей, что соответствуют полям типа foreignkey, присутствующим в перечне из атрибута List\_display;

• список или кортеж, содержащий строки с именами полей типа foreignkey, — извлекать связанные записи только тех первичных моделей, что соответствуют приведенным в этом списке (кортеже) полям;

• ’’пустой” список или кортеж — вообще не извлекать связанные записи.

Извлечение связанных записей первичных моделей одновременно с записями текущей модели повышает производительность, т. К. Впоследствии Django не придется для получения связанных записей обращаться к базе еще раз.

Значение по умолчанию — False;

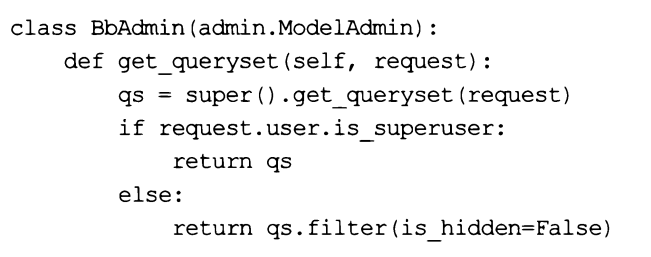
□ get\_list\_select\_related(self, request) — метод, должен возвращать набор первичных моделей, чьи связанные записи будут извлекаться одновременно с записями текущей модели посредством вызова метода select\_related (). Полученный с параметром request запрос можно использовать при формировании этого набора. В изначальной реализации возвращает значение атрибута:

****

□ get queryset (seif, request) — метод, должен возвращать набор записей, который и станет выводиться на странице списка. Параметр request передает запрос.

Переопределив этот метод, можно установить какую-либо дополнительную фильтрацию записей, создать вычисляемые поля или изменить сортировку.

Вот пример переопределения этого метода с целью дать возможность суперпользователю просматривать все объявления, а остальным пользователям — только объявления, не помеченные как скрытые:

****

**Параметры списка записей: фильтрация и сортировка**

В этом разделе перечислены параметры, настраивающие средства для фильтрации и сортировки записей:

□ ordering — атрибут, задает порядок сортировки записей. Значение указывается в том же формате, что и значение параметра модели ordering.

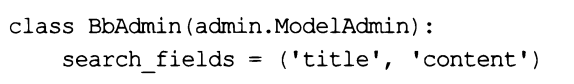
Если указать значение None, то будет использована сортировка, заданная для модели. По умолчанию — None;

□ get\_ordering(self, request) — метод, должен возвращать параметры сортировки. Параметр request передает запрос. В изначальной реализации возвращает значение атрибута ordering;

□ sortabie by — атрибут, задает перечень полей модели, по которым пользователь сможет выполнять сортировку записей. Перечень полей задается в виде списка, кортежа или множества. Если задать ’’пустую” последовательность, то пользователь не сможет сортировать записи по своему усмотрению. По умолчанию — None (разрешена сортировка записей по всем полям);

□ get sortabie by(self, request) — метод, должен возвращать перечень полей модели, по которым пользователь сможет сортировать записи. Параметр request передает запрос. В изначальной реализации возвращает значение атрибута Sortable\_by;

□ search fields — атрибут, указывает перечень полей модели, по которым будет выполняться фильтрация записей. Значением должен быть список или кортеж имен полей, заданных в виде строк. Пример:

****

Фильтрация по указанным полям осуществляется занесением в расположенное вверху страницы поле ввода искомого слова или фразы и нажатием кнопки найти. Чтобы отменить фильтрацию и вывести все записи, достаточно очистить поле ввода и снова нажать кнопку Найти.

По умолчанию Django выбирает записи, которые содержат все занесенные в поле ввода слова, независимо от того, в каком месте поля встретилось это слово и является оно отдельным словом или частью другого, более длинного. Фильтрация выполняется без учета регистра. Например, если ввести фразу "газ кирпич", будут отобраны записи, которые в указанных полях хранят слова "Газ" и "Кирпич", "керогаз" и "кирпичный" и т. П., но не "бензин" и "дерево".

Чтобы изменить поведение фреймворка при фильтрации, нужно предварить имя поля одним из поддерживаемых префиксов:

• А — искомое слово должно присутствовать в начале поля: Search—fields = (’Acontent’,) при задании слова "газ" будут отобраны записи со словами "газ", "газовый", но не "керогаз";

• = — точное совпадение, т. Е. Указанное слово должно полностью совпадать со значением поля. Регистр не учитывается. Пример: Search—fields = (’=content’,) будут отобраны записи со словами "газ", но не "газовый" или "керогаз";

• @ — полнотекстовый поиск. Поддерживается только базами данных mysql.

Значение по умолчанию — "пустой" кортеж;

□ get—search—fields (seif, request) — метод, должен возвращать перечень полей, по которым будет выполняться фильтрация записей. Параметр request передает запрос. В изначальной реализации возвращает значение атрибута search\_fieids;

□ show full result count — атрибут. Если True, то после выполнения фильтрации в полях, заданных атрибутом search fieids, правее кнопки Найти будет выведено количество отфильтрованных записей и общее число записей в модели.

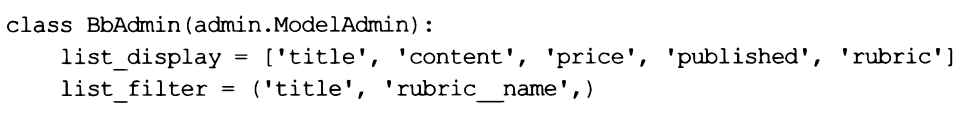
Если False, то вместо общего числа записей в модели будет выведена гиперссылка показать все. По умолчанию — True.

Для вывода общего количества записей Django выполняет дополнительный запрос к базе данных. Поэтому, если стоит задача всемерно уменьшить нагрузку на базу, этому атрибуту имеет смысл задать значение False;

□ list filter — атрибут, указывает перечень полей, по которым можно будет выполнять быструю фильтрацию. В правой части страницы списка записей появятся списки всех значений, занесенных в заданные поля, и при щелчке на таком значении будут выведены только те записи, у которых указанное поле хранит выбранное значение. Чтобы вновь вывести в списке все записи, следует щелкнуть пункт Все.

Значением поля должен быть список или кортеж, каждый элемент которого представляет собой:

• имя поля в виде строки:

****

Как видно из примера, в перечне можно указывать поля связанных моделей;

• ссылку на подкласс класса simplelistfilter ИЗ модуля django. Contrib.admin, реализующий более сложную фильтрацию. В этом подклассе следует объявить:

° title — атрибут, указывает заголовок, который будет выведен над перечнем доступных для выбора значений, по которым собственно и будет выполняться фильтрация. Значение задается в виде строки;

D parameter name — атрибут, указывает имя GET-параметра, посредством которого будет пересылаться внутренний идентификатор выбранного пользователем значения для фильтрации. Это имя также должно представлять собой строку;

О lookups (self, request, model\_admin) — метод, должен возвращать перечень доступных для выбора значений, по которым будет выполняться фильтрация. Через параметр request передается объект запроса, через параметр model admin — объект редактора.

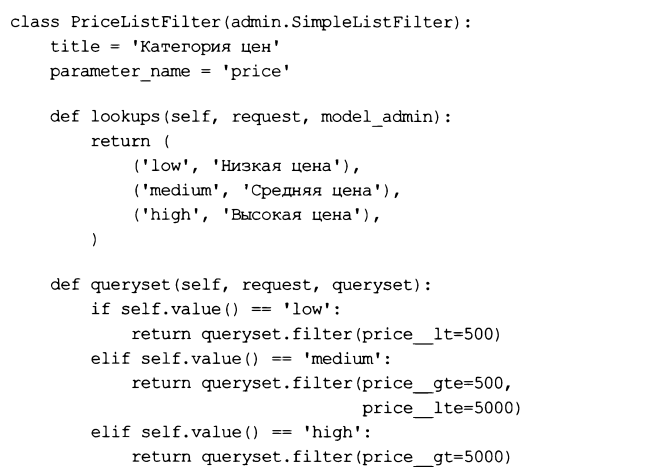
Метод должен возвращать кортеж, каждый элемент которого задаст отдельное значение для фильтрации. Этот элемент должен представлять собой кортеж из двух строковых элементов: внутреннего идентификатора значения (того самого, что пересылается через GET-параметр, заданный атрибутом parameter name) и названия, выводящегося на экран;

D queryset (self, request, queryset) — метод, вызываемый после щелчка пользователя на одном из значений и возвращающий соответствующим образом отфильтрованный набор записей. Через параметр request передается объект запроса, через параметр queryset — изначальный набор записей.

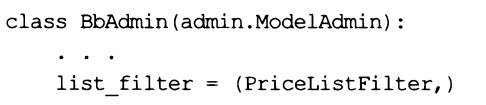
Чтобы получить внутренний идентификатор значения, на котором щелкнул пользователь, нужно обратиться к методу value (), унаследованному классом редактора от суперкласса.

Далее приведен код класса, реализующего фильтрацию объявлений по цене:

Является она низкой (менее 500 руб.), средней (от 500 до 5000 руб.) Или высокой (более 5000 руб.).

****

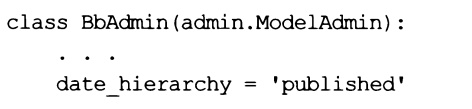
Теперь мы можем использовать этот класс для быстрой фильтрации объявлений по категории цен:



Если в качестве значения атрибута указать ’’пустой” список или кортеж, то быстрая сортировка будет отключена. Значение по умолчанию — ’’пустой” кортеж;

□ get iist fiiter(seif, request) — метод, должен возвращать перечень полей, по которым будет выполняться быстрая фильтрация. Параметр request передает запрос. В изначальной реализации возвращает значение атрибута list filter;

□ date hierarchy— атрибут, включающий быструю фильтрацию по датам или временным отметкам. Если указать в качестве его значения строку с именем поля типа datefield или datetimefield, то над списком записей будет выведен набор значений, хранящихся в этом поле, в виде гиперссылок. После щелчка на такой гиперссылке в списке будут выведены только те записи, в указанном поле которых хранится выбранное значение. Пример:



Значение по умолчанию — None (быстрая фильтрация по дате не выполняется);

□ preserve filters— атрибут. Если True, то после сохранения добавленной или исправленной записи все заданные пользователем условия фильтрации продолжают действовать. Если False, то фильтрация записей в списке после этого будет отменена. По умолчанию — True.

Параметры списка записей: прочие описанные здесь параметры затрагивают по большей части внешний вид списка записей:

□ list per page — атрибут, задает количество записей в части пагинатора, применяемого при выводе списка записей, в виде целого числа (по умолчанию: юо);

□ list max show aii— атрибут, задает количество записей, выводимых после щелчка на гиперссылке показать все (она находится под списком), если общее количество записей меньше или равно количеству, заданному этим атрибутом.

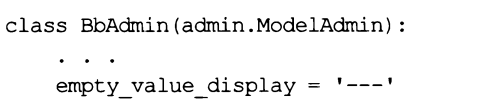
Значение атрибута указывается в виде целого числа. По умолчанию: 200;

□ actions on top— атрибут. Если True, то над списком записей будет выведен раскрывающийся список всех действий, доступных для выполнения над группой выбранных записей. Если False, то список действий над списком записей выведен не будет. По умолчанию — True;

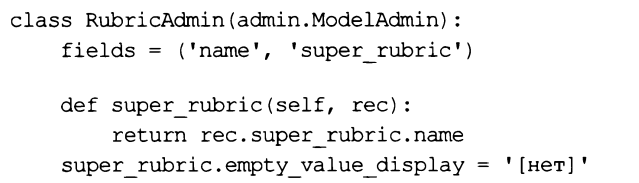
□ actions on bottom— атрибут. Если True, то под списком записей будет выведен раскрывающийся список всех действий, доступных для выполнения над группой выбранных записей. Если False, то список действий под списком записей выведен не будет. По умолчанию — False;

□ actions selection counter— атрибут. ЕСЛИ True, ТО возле раскрывающегося списка действий будут выведены количество выбранных записей и общее количество записей на текущей странице списка. Если False, то эти сведения не будут ВЫВОДИТЬСЯ. По умолчанию — True;

□ empty vaiue display— атрибут, указывает символ или строку, которая будет выводиться вместо пустого значения поля (таким считается ’’пустая” строка и None). По умолчанию: (дефис). Пример:

****

Если в редакторе объявлено функциональное поле, то можно указать подобного рода значение только для этого поля, присвоив его атрибуту empty vaiue display объекта метода, который реализует это поле:

****

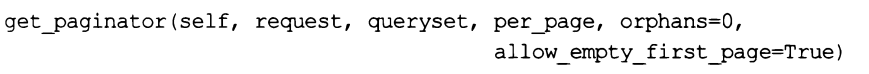
Наконец, можно указать ’’замещающее” значение для всего административного

Сайта:



□ paginator— атрибут, задает ссылку на класс пагинатора, используемого для вывода списка записей. По умолчанию — ссылка на класс Paginator из модуля Dj ango.core.paginator;

□ get paginator () — метод, должен возвращать объект пагинатора, используемого для вывода списка записей. Формат объявления:

****

Параметр request передает запрос, параметр queryset — набор записей, который должен разбиваться на части, параметр per page — количество записей в части.

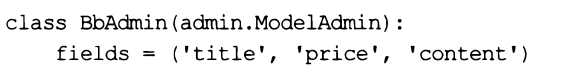
В реализации по умолчанию возвращает экземпляр класса пагинатора, заданного в атрибуте paginator.

**Параметры страниц добавления и правки записей: набор выводимых полей**

Эти параметры указывают набор выводимых в форме полей модели, перечни наборов полей (будут описаны далее) и некоторые другие параметры:

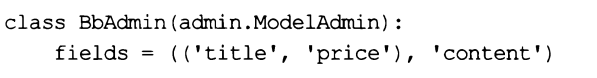
□ fields — атрибут, задает последовательность (список или кортеж) имен полей, выводимых в форме (не указанные поля выведены не будут). Поля будут выведены в том порядке, в котором они перечислены в последовательности.

Пример вывода в форме объявления только полей названия, цены и описания (именно в таком порядке):



В наборе можно указать поля, доступные для чтения, которые указаны в атрибуте readoniy fieids (он будет описан далее).

По умолчанию поля в форме выводятся по вертикали сверху вниз. Чтобы вывести какие-либо поля в одной строчке по горизонтали, нужно заключить их имена во вложенный кортеж. Пример вывода полей названия и цены в одной строчке:

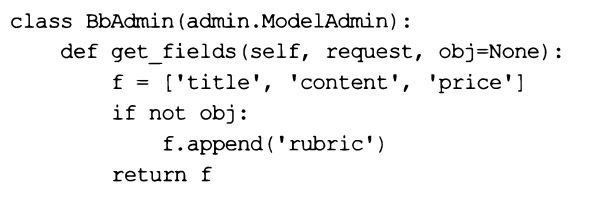


Если указать для атрибута значение None, то в форме будут выведены все поля, кроме имеющих ТИПЫ autofield, smallautofield, bigautofield И тех, у которых в параметре editable конструктора класса было явно или неявно задано значение True.

Значение по умолчанию — None;

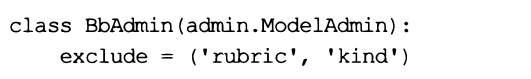
□ get fields (seif, request, obj=None) — метод, должен возвращать последовательность имен полей, которые следует вывести в форме. В параметре request передается запрос, а в необязательном параметре obj может быть получен объект исправляемой записи или None, если запись в данный момент добавляется.

Для примера сделаем так, чтобы у создаваемого объявления можно было указать рубрику, а у исправляемого — уже нет:

****

В изначальной реализации метод возвращает значение атрибута fields, если его значение отлично от None. В противном случае возвращается список, включающий все поля модели и все поля, доступные только для чтения (указываются в атрибуте readoniy\_fieids, описанном далее);

□ exclude — атрибут, задает последовательность имен полей, наоборот, не выводимых в форме (не указанные поля будут выведены). Пример вывода в форме объявления всех полей, кроме рубрики и типа объявления:



Если задать значение None, то ни одно поле модели не будет исключено из числа выводимых в форме. Значение по умолчанию — None;

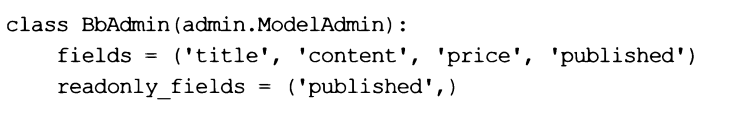
□ get\_exciude (seif, request, obj=None) — метод, должен возвращать последовательность имен полей модели, не выводимых в форме. В параметре request передается запрос, а в необязательном параметре obj может быть получен объект исправляемой записи или None, если запись в данный момент добавляется.

В изначальной реализации возвращает значение атрибута exclude;

Внимание!

Следует задавать либо перечень выводимых в форме полей, либо перечень не выводимых полей. Если указать их оба, возникнет ошибка.

□ readoniy fieids— атрибут, задает последовательность имен полей, которые должны быть доступны только для чтения. Значения таких полей будут выведены в виде обычного текста. Пример:

****

Указание в списке атрибута readoniy\_fieids — единственный способ вывести на странице правки записи значение поля, у которого при создании параметр Editable конструктора был установлен В False.

Также в этом атрибуте можно указать функциональные поля, объявленные в классе редактора.

Значение по умолчанию — ’’пустой” кортеж;

□ get\_readonly\_fields (self, request, obj=None) — метод, должен возвращать перечень полей, которые должны быть доступны только для чтения. В параметре Request передается запрос, а в необязательном параметре obj может быть получен объект исправляемой записи или None, если запись в данный момент добавляется. В изначальной реализации возвращает значение атрибута readonlyfields;

□ inlines— атрибут, задает список встроенных редакторов, присутствующих в текущем редакторе (будут рассмотрены позже). По умолчанию— ’’пустой” список;

□ fieldsets — атрибут, задает перечень наборов полей, которые будут созданы в форме.

Набор полей, как и следует из его названия, объединяет указанные поля формы.

Он может вообще никак не выделяться на экране (основной набор полей), а может представляться в виде спойлера, изначально свернутого или развернутого.

Перечень наборов полей записывается в виде кортежа, в котором каждый элемент представляет один набор и также должен являться кортежем из двух элементов: заголовка набора полей (если задать None, будет создан основной набор полей) и словаря со следующими дополнительными параметрами:

• fields — кортеж из строк с именами полей модели, которые должны выводиться в наборе. Этот параметр обязателен для указания.

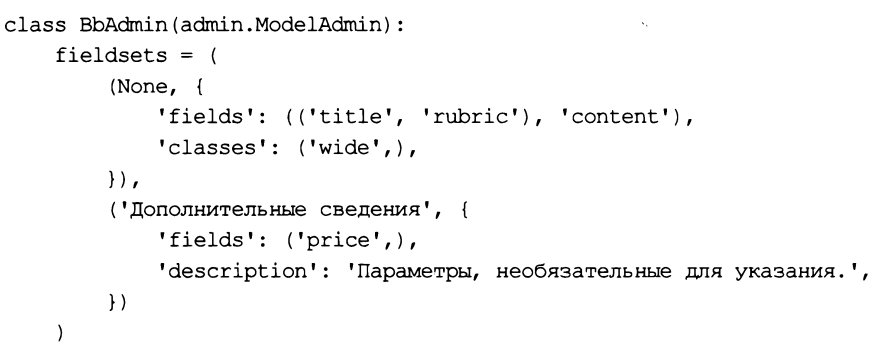
Здесь можно указать также доступные для чтения поля, заданные в атрибуте Readoniy fieids, в том числе и функциональные поля.

По умолчанию поля в наборе выводятся по вертикали сверху вниз. Чтобы вывести какие-либо поля в одной строчке по горизонтали, нужно заключить их имена во вложенный кортеж;

• classes — список или кортеж с именами стилевых классов, которые будут привязаны к набору, представленными строками. Поддерживаются два стилевых класса: collapse (набор полей, представленный в виде спойлера, будет изначально свернут) и wide (поля в наборе займут все пространство окна по ширине);

• description — строка с поясняющим текстом, который будет выведен вверху набора форм, непосредственно под его названием. Вместо обычного текста можно указать HTML-код.

Пример указания набора полей:

****

□ get\_fieldsets (self, request, obj=None) — метод, должен возвращать перечень наборов полей, которые будут выведены в форме. В параметре request передается запрос, а в необязательном параметре obj может быть получен объект исправляемой записи или None, если запись в данный момент добавляется.

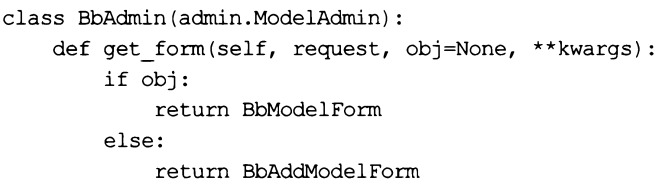
В изначальной реализации возвращает значение атрибута fieldsets, если его значение отлично от ’’пустого” словаря. В противном случае возвращается перечень из одного основного набора, содержащего все имеющиеся в форме поля;

□ form— атрибут, задает класс связанной с моделью формы, на основе которой будет создана окончательная форма, применяемая для работы с записью. Значение ПО умолчанию — ссылка на класс modeiform;

□ get\_form(self, request, obj=None, \*\*kwargs) — метод, должен возвращать класс формы, которая будет использоваться для занесения данных в создаваемую или исправляемую запись.

В параметре request передается запрос, в необязательном параметре obj может быть получен объект исправляемой записи или None, если запись в данный момент добавляется. Параметр kwargs хранит словарь, элементы которого будут переданы функции modeiform factoryo, применяемой методом для создания класса формы, в качестве дополнительных параметров.

Вот пример использования ДЛЯ добавления записи формы bbaddmodelform, а для правки существующей записи---формы bbmodelform:

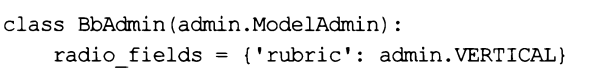
****

В изначальной реализации метод возвращает класс формы, сгенерированный ВЫЗОВОМ функции model fonn factory ().

**Параметры страниц добавления и правки записей: Элементы управления**

Эти параметры настраивают внешнее представление отдельных полей в форме, в частности, задают для них элементы управления:

□ radio\_fieids — атрибут, задает перечень полей типа foreignkey и полей со списком, для отображения которых вместо раскрывающегося списка будет использован набор переключателей. В качестве значения задается словарь, ключами элементов которого должны быть имена полей, а значениями — одна из переменных, объявленных в модуле django. Contrib. Admin: HORIZONTAL (переключатели в наборе располагаются по горизонтали) или vertical (по вертикали). Пример:

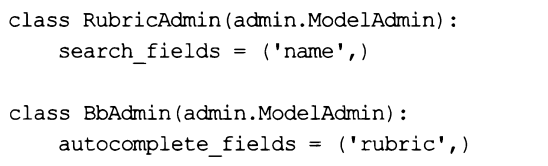


Значение по умолчанию — ’’пустой” словарь;

□ autocomplete\_fieids— атрибут, задает перечень полей ТИПОВ foreignkey и Manytomanyfieid, которые должны отображаться на экране в виде списка с возможностью поиска. Такой список будет включать в свой состав поле для указания текста искомого пункта или его части. Значение атрибута указывается в виде списка или кортежа, включающего строки с именами полей.

Внимание!

Чтобы список с возможностью поиска успешно работал, необходимо в классе редактора, представляющем связанную модель, задать перечень полей, по которым можно ВЫПОЛНЯТЬ ПОИСК записей (атрибут search\_fieids). Пример:

****

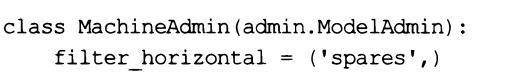
Значение по умолчанию — ’’пустой” кортеж;

□ get autocomplete fieids (self, request) — метод, должен возвращать перечень полей, отображаемых в виде списка с возможностью поиска. В параметре Request передается запрос. В изначальной реализации возвращает значение атрибута autocomplete\_fieids;

□ filter horizontai— атрибут, указывает кортеж строк с именами полей типа Manytomanyfieid, которые должны быть отображены в виде пары расположенных по горизонтали списков с возможностью поиска пунктов.

Пример такого элемента управления, надо сказать, исключительно удобного в использовании, можно увидеть на рис. 15.1. В левом списке выводятся только записи ведомой модели, не связанные с текущей записью ведущей модели, а в правом— только записи, связанные с ней. Связывание записей ведомой модели с текущей записью ведущей модели выполняется переносом их из левого списка в правый щелчком на кнопке со стрелкой, направленной вправо. Аналогично удаление записей из числа связанных с текущей записью производится переносом из правого списка в левый, для чего нужно щелкнуть на кнопке со стрелкой влево.

Вот пример вывода ПОЛЯ spares модели Machine в виде подобного рода элемента управления:

****

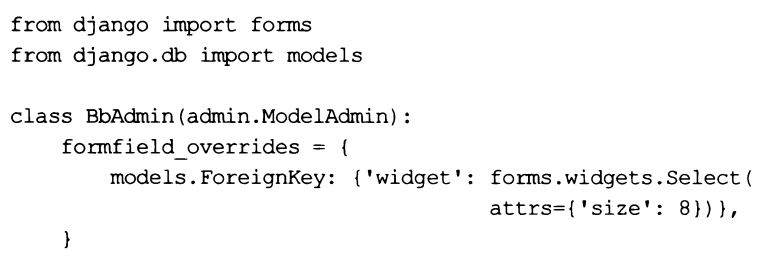
Поля типа manytomanyfieid, не указанные в этом атрибуте, будут выводиться в виде обычного списка с возможностью выбора произвольного количества пунктов. Такой элемент управления не очень удобен в использовании, особенно если записей в ведомой модели достаточно много.

Значение по умолчанию — ’’пустой” кортеж;

□ filter vertical — ТО же самое, ЧТО filter horizontal, ТОЛЬКО СПИСКИ ВЫВОДЯТСЯ не по горизонтали, а по вертикали, друг над другом: сверху — список несвязанных записей, а под ним — список связанных записей;

□ formfield overrides— атрибут, позволяет переопределить параметры полей формы, которая будет выводиться на страницах добавления и правки. В качестве значения указывается словарь, ключами элементов которого выступают ссылки на классы полей модели, а значения указывают параметры соответствующих им полей формы и также записываются в виде словарей.

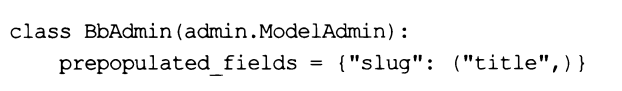
Чаще всего этот атрибут применяется для задания других элементов управления для полей формы. Вот пример указания для поля типа foreignkey в качестве элемента управления обычного списка вместо применяемого по умолчанию раскрывающегося:

****

Значение по умолчанию — ’’пустой” словарь;

□ prepopuiated fieids — атрибут, устанавливает набор полей, значения которых должны формироваться на основе значений из других полей. В качестве значения указывается словарь, ключи элементов которого должны соответствовать полям, значения которых будут формироваться описанным ранее образом, а значениями станут кортежи имен полей, откуда будут браться данные для формирования значений.

Основное назначение этого атрибута— указание сведений для формирования слагов. Обычно слаг создается из названия какой-либо позиции путем преобразования букв кириллицы в символы латиницы, удаления знаков препинания и замены пробелов на дефисы. Также можно формировать слаги на основе нескольких значений (например, названия и рубрики) — в этом случае отдельные значения объединяются. Пример:



Значение по умолчанию — ’’пустой” словарь;

□ get\_prepopulated\_fieids (self, request, obj=None)—метод, должен возвращать набор полей, значения которых должны формироваться на основе значений из других полей. В параметре request передается запрос, в параметре obj — объект исправляемой записи или None, если запись в данный момент добавляется. В изначальной реализации возвращает значение атрибута prepopuiated\_fieids;

□ raw id fields — атрибут, задает перечень полей типа foreignkey или Manytomanyfieid, для отображения которых вместо списка нужно использовать обычное поле ввода. В такое поле вводится ключ связанной записи или сразу несколько ключей через запятую. Значение атрибута указывается в виде списка или кортежа, в котором приводятся строки с именами полей. Значение по умолчанию — ’’пустой” кортеж.

**Параметры страниц добавления и правки записей: Прочие**

Эти, весьма немногочисленные, параметры управляют внешним видом и поведением страниц добавления и правки записей:

□ view on site— атрибут, указывает, будет ли на странице правки записи выводиться гиперссылка Смотреть на сайте, при щелчке на которой осуществляется переход по интернет-адресу модели. В качестве значения атрибута можно указать:

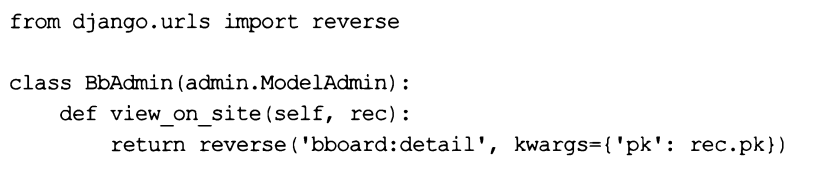
• True — ВЫВОДИТЬ гиперссылку, если В модели Объявлен метод get\_absolute\_Urlо, формирующий интернет-адрес модели. Если такого метода нет, то гиперссылка выводиться не будет;

• False — не выводить гиперссылку в любом случае.

Значение по умолчанию — True.

Аналогичного результата можно достичь, объявив непосредственно в классе редактора метод view\_on\_site(), который в качестве единственного параметра будет получать объект записи и возвращать строку с интернет-адресом модели.

Вот пример:

****

□ save as — атрибут. Если True, то на странице будет выведена кнопка Сохранить как новый объект, если False, то вместо нее будет присутствовать кнопка сохранить и добавить другой объект. По умолчанию — False;

□ save as continue— атрибут. Принимается во внимание только в том случае, если для атрибута save as задано значение True. Если True, то после сохранения новой записи выполняется перенаправление на страницу правки той же записи, если False — возврат на страницу списка записей. По умолчанию — True;

□ save on top — атрибут. Если True, кнопки сохранения записи будут присутствовать И вверху, И внизу страницы, если False — только внизу. По умолчанию — False.

На заметку!

Помимо рассмотренных здесь, редакторы Django поддерживают ряд более развитых инструментов: обработку сохранения записей и наборов записей, удаления записей, указание элементов управления для полей в зависимости от различных условий (например, является ли текущий пользователь суперпользователем), замену шаблонов страниц и др. Эти инструменты достаточно сложны в использовании и применяются относительно редко. За инструкциями по

Их применению обращайтесь на страницу https://docs.djangoproject.eom/en/3.0/ref/Contrib/admin/.

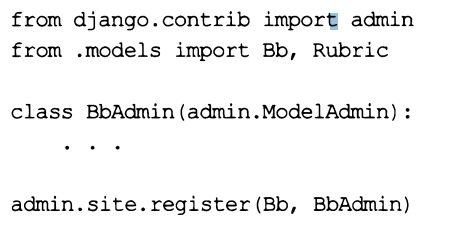
**Регистрация редакторов на административном веб-сайте**

Чтобы административный сайт смог использовать в работе редактор, последний должен быть соответствующим образом зарегистрирован. Сделать это можно двумя способами:

□ применить расширенный формат вызова метода register о объекта административного сайта:



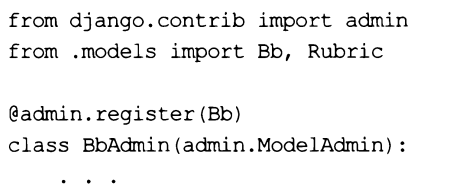
Пример:

****

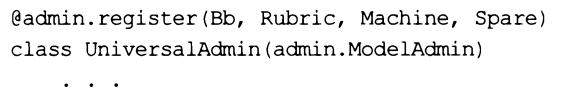
□ применить декоратор register о, объявленный в модуле django. Contrib. Admin, формат вызова которого следующий:



Этот декоратор указывается непосредственно у объявления класса редактора. Пример:

****

Применив декоратор, один и тот же класс редактора можно указать сразу для нескольких моделей:

****

1. Встроенные редакторы

Встроенный редактор по назначению аналогичен встроенному набору форм. Он создает на странице добавления или правки записи первичной модели набор форм для работы со связанными записями вторичной модели.

**Объявление встроенного редактора**

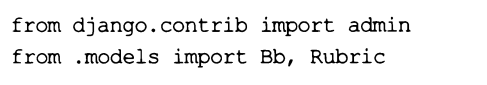
Класс встроенного редактора должен быть производным от одного из следующих классов, объявленных В модуле django.contrib.admin:

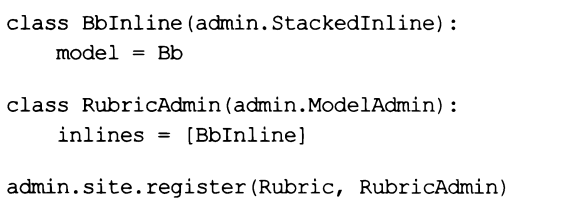
□ stackediniine — элементы управления располагаются по вертикали;

□ Tabuiariniine — элементы управления располагаются по горизонтали. Для формирования набора форм применяется таблица HTML.

В объявлении встроенного редактора указывается модель, которую он должен обслуживать. После объявления он связывается с классом основного редактора.

Приведен КОД, объявляющий и регистрирующий редактор rubricadmin, который предназначен для работы с моделью рубрик Rubric. Этот редактор связан со встроенным редактором Bbiniine, обслуживающим модель объявлений вь и выводящим объявления из текущей рубрики.

****

****

**Параметры встроенного редактора**

Оба класса встроенных редакторов наследуют от класса modeladmin атрибуты Ordering, fields, exclude, fieldsets, radio\_fields, filter\_horizontal, filtervertical, formfield\_overrides, readonly\_fields, prepopulated\_fieids, raw\_id\_fields и методы get\_ordering (), get\_queryset (), get\_fieids (), get\_exclude (), get\_Fieldsets (), get\_readonly\_fieids (), get\_prepopulated\_fieids (), описанные ранее.

Кроме того, встроенные редакторы поддерживают следующие дополнительные атрибуты и методы:

□ model — атрибут, указывает ссылку на класс вторичной модели, которая будет обслуживаться встроенным редактором. Единственный обязательный для указания атрибут. По умолчанию — None;

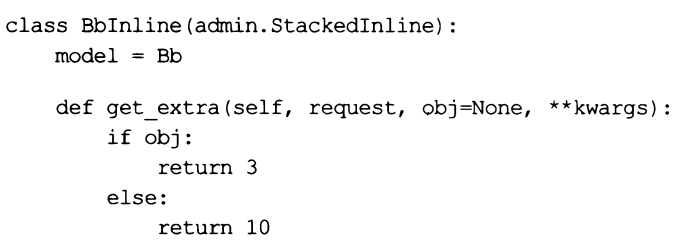
□ fk name — атрибут, задает имя поля внешнего ключа у вторичной модели в виде строки. Обязателен для указания, если вторичная модель связана с разными первичными моделями и, соответственно, включает несколько полей внешнего ключа. Если задать значение None, то Django использует самое первое из объявленных в модели поле внешнего ключа. По умолчанию — None;

□ extra —атрибут, указывает количество пустых форм, предназначенных для создания новых записей, которые будут присутствовать в редакторе. Количество форм должно быть задано в виде целого числа. По умолчанию: з;

□ get\_extra(self, request, obj=None, \*\*kwargs) — метод, должен возвращать количество пустых форм, предназначенных для создания новых записей.

В параметре request передается запрос, в параметре obj — объект исправляемой записи первичной модели или None, если запись еще не создана.

Вот пример указания десяти ’’пустых” форм при создании записи первичной модели и трех при ее правке:



В изначальной реализации метод возвращает значение атрибута extra;

□ can deiete — атрибут. Если True, то редактор разрешит пользователю удалять записи, если False — не разрешит. По умолчанию — True;

□ show change link— атрибут. Если True, то в каждой из форм встроенного редактора будет выведена гиперссылка изменить, ведущая на страницу правки соответствующей записи. Если False, то такая гиперссылка выводиться не будет. По умолчанию — False;

□ min num— атрибут, указывает минимальное допустимое количество форм в редакторе в виде целого числа. Если задать None, редактор может включать сколько угодно форм. По умолчанию — None;

□ get\_min\_num(self, request, obj=None, \*\*kwargs) — метод, должен возвращать минимальное допустимое количество форм в редакторе. В параметре request передается запрос, в параметре obj — объект исправляемой записи первичной модели или None, если запись еще не создана.

В изначальной реализации метод возвращает значение атрибута min num;

□ max num— атрибут, указывает максимальное допустимое количество форм в редакторе в виде целого числа. Если задать None, то редактор может включать СКОЛЬКО угодно форм. По умолчанию — None;

□ get\_max\_num(self, request, obj=None, \*\*kwargs) — метод, должен возвращать максимальное допустимое количество форм в редакторе. В параметре request передается запрос, в параметре obj — объект исправляемой записи первичной модели или None, если запись еще не создана.

В изначальной реализации метод возвращает значение атрибута max num;

□ classes — атрибут, задает набор стилевых классов, которые будут привязаны к встроенному редактору. Значение указывается в виде списка или кортежа, содержащего строки с именами стилевых классов. Поддерживаются два стилевых класса: collapse (редактор, представленный в виде спойлера, будет изначально свернут) и wide (редактор займет все пространство окна по ширине). По умолчанию — None;

□ verbose name — атрибут, задает название сущности, хранящейся в записи вторичной модели, в виде строки. Если задать значение None, то будет использовано название, записанное в одноименном параметре обслуживаемой редактором модели. По умолчанию — None;

□ verbose name piurai — атрибут, задает название набора сущностей, хранящихся во вторичной модели, в виде строки. Если задать значение None, то будет использовано название, записанное в одноименном параметре обслуживаемой редактором модели. По умолчанию — None;

□ formset — атрибут, указывает набор форм, связанный с моделью, на основе которого будет создан окончательный набор форм, выводимый на страницу. По умолчанию — ссылка на класс baselnlineformset;

□ get\_fоrmset (self, request, obj=None, \*\* kwargs) —метод, должен возвращать класс встроенного набора форм, который будет использован в редакторе.

В параметре request передается запрос, в параметре obj — объект исправляемой записи первичной модели или None, если запись еще не создана. Параметр kwargs хранит словарь, элементы которого будут переданы функции iniineformset\_Factory (), создающей класс набора форм, в качестве параметров.

В изначальной реализации метод возвращает класс встроенного набора форм, сгенерированный ВЫЗОВОМ функции inlineformset\_factory ();

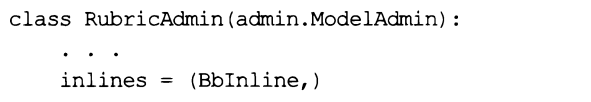
□ form— атрибут, задает класс связанной с моделью формы, которая будет применяться в создаваемом наборе форм. По умолчанию — ссылка на класс Modelform.

**Регистрация встроенного редактора**

Для регистрации встроенных редакторов в редакторе, обслуживающем первичную модель, могут быть использованы:

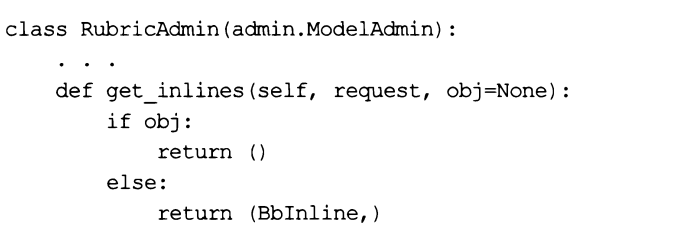
□ inlines — атрибут, задает список или кортеж со ссылками на классы встроенных редакторов, регистрируемых в текущем редакторе. По умолчанию — ’’пустой” список.

Регистрация встроенного редактора Bbiniine, обслуживающего вторичную модель вь, в редакторе rubricadmin первичной модели:



□ get iniines (self, request, obj) (начиная c Django 3.0)— метод, должен возвращать последовательность ссылок на классы регистрируемых встроенных редакторов. В параметре request передается запрос, в параметре obj — объект исправляемой записи первичной модели или None, если такой записи еще нет.

Пример вывода набора форм для ввода объявлений только на странице добавления рубрики (на странице правки рубрики он выводиться не будет):



В изначальной реализации метод возвращает значение атрибута inlines.

1. Действия

Действие в терминологии административного сайта Django — это операция, выбираемая из раскрывающегося списка Действие и выполняемая применительно к выбранным в списке записям. Список Действие находится над перечнем записей, а увидеть его можно на рис. 1.8 и 1.10.

Изначально в этом списке присутствует лишь действие Удалить выбранные Название набора сущностей^, однако можно добавить туда свои собственные действия.

Сначала необходимо объявить функцию, которая, собственно, и реализует действие. В качестве параметров она должна принимать:

□ экземпляр класса редактора, к которому будет привязано действие;

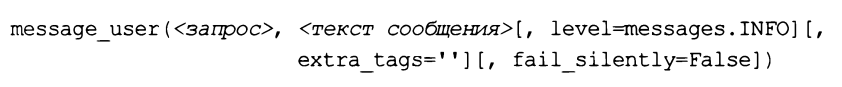
□ запрос;

□ набор записей, содержащий записи, которые были выбраны пользователем.

Никакого результата она возвращать не должна.

Далее нужно задать для действия название, которое будет выводиться в списке действие. Строка с названием присваивается атрибуту short description объекта функции, реализующей это действие.

По завершении выполнения действия, равно как и при возникновении ошибки, рекомендуется вывести всплывающее сообщение. Делается это вызовом метода message\_user () класса modeladmin:



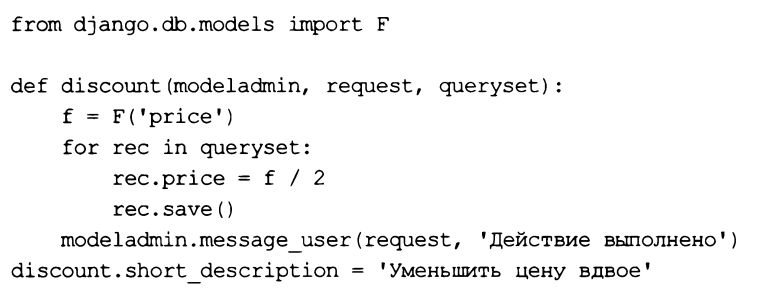
Запрос представляется экземпляром класса httprequest, текст сообщения—строкой.

Необязательный параметр level указывает уровень сообщения.

Необязательный параметр extra tags задает перечень дополнительных стилевых классов, привязываемых к HTML-тегу с текстом выводимого всплывающего сообщения. Параметру присваивается строка со стилевыми классами, разделенными пробелами.

Если присвоить необязательному параметру faii silentiy значение True, то в случае невозможности создания нового всплывающего сообщения (например, если соответствующая подсистема отключена) ничего не произойдет. Используемое по умолчанию значение False указывает в таком случае возбудить исключение Messagefailure ИЗ модуля django. Cont rib. Messages.

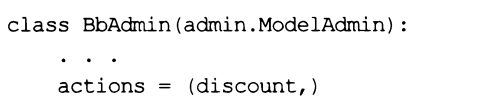
Приведен код функции discount (), реализующей действие, которое уменьшит цены в выбранных объявлениях вдвое и уведомит о завершении всплывающим сообщением.

****

Для регистрации действий в редакторе предусмотрен атрибут actions, поддерживаемый классом modeladmin. Атрибуту присваивается список или кортеж, содержащий ссылки на функции, что реализуют регистрируемые в редакторе действия.

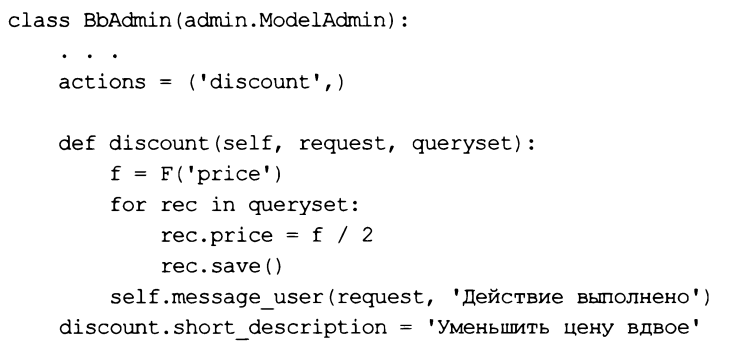
Значение атрибута actions по умолчанию — ’’пустой” список.

Вот так в редакторе bbadmin регистрируется действие discount ():



Действие можно реализовать в виде метода того же класса редактора, в котором оно будет зарегистрировано. Имя метода, реализующего действие, в списке атрибута actions следует указать в виде строки.

Пример действия, реализованного в виде метода discount () редактора bbadmin:

****

На заметку!

Django поддерживает создание более сложных действий, выводящих какие-либо промежуточные страницы (наподобие страницы подтверждения), доступных во всех редакторах, а также временную деактивацию действий. Соответствующие руководства находятся здесь: https://docs.djangoprojectcom/en/3.0/ref/contrib/admin/actions/.