**Тема занятия № 17: Django Формы**

1. Создание форм, связанных с моделями

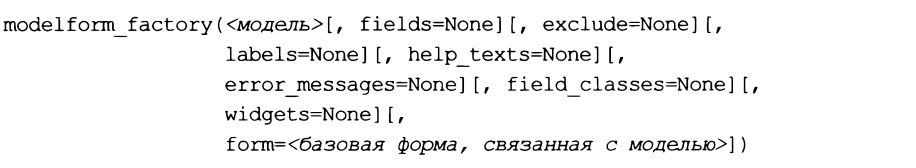
**Форма** в терминологии Django — это объект, выводящий на страницу веб-форму для занесения данных и проверяющий введенные данные на корректность. Форма определяет набор полей, в которые будут вводиться отдельные значения, типы заносимых в них значений, элементы управления, посредством которых будет осуществляться ввод данных, и правила валидации.

Форма, связанная с моделью, отличается от обычной формы тем, что представляет какую-либо запись модели — хранящуюся в базе данных или еще не существующую. В частности, поля такой формы соответствуют одноименным полям модели.

Помимо этого, такая форма поддерживает метод save (), сохраняющий занесенные в форму данные в базе.

Существуют три способа создать форму, связанную с моделью: два простых и сложный.

Создание форм с помощью фабрики классов первый, самый простой способ создать форму, связанную с моделью, — использовать функцию modelform factory () ИЗ МОДУЛЯ django. Forms. Вот формат ее вызова:

****

В первом параметре указывается ссылка на класс модели, на основе которой нужно создать форму.

Параметр fields задает последовательность имен полей модели, которые должны быть включены в создаваемую форму. Любые поля, не включенные в эту последовательность, не войдут в состав формы. Чтобы указать все поля модели, нужно присвоить этому параметру строку "\_\_all\_\_".

Параметр exclude задает последовательность имен полей модели, которые, напротив, не должны включаться в форму. Соответственно, все поля, отсутствующие в этой последовательности, войдут в состав формы.

Внимание!

В вызове функции modeiform factoryo должен присутствовать либо параметр fields, либо параметр exclude. Указание сразу обоих параметров приведет к ошибке.

Параметр labels задает надписи для полей формы. Его значение должно представлять собой словарь, ключи элементов которого соответствуют полям формы, а значения задают надписи для них.

Параметр help texts указывает дополнительные текстовые пояснения для полей формы (такой текст будет выводиться возле элементов управления). Значение этого параметра должно представлять собой словарь, ключи элементов которого соответствуют полям формы, а значения задают пояснения.

Параметр error messages указывает сообщения об ошибках. Его значением должен быть словарь, 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.

Параметр fieid ciasses указывает, поле какого типа должно быть создано в форме для соответствующего ему поля модели. Значением должен быть словарь, ключи элементов которого представляют имена полей модели, а значениями элементов станут ссылки на соответствующие им классы полей формы.

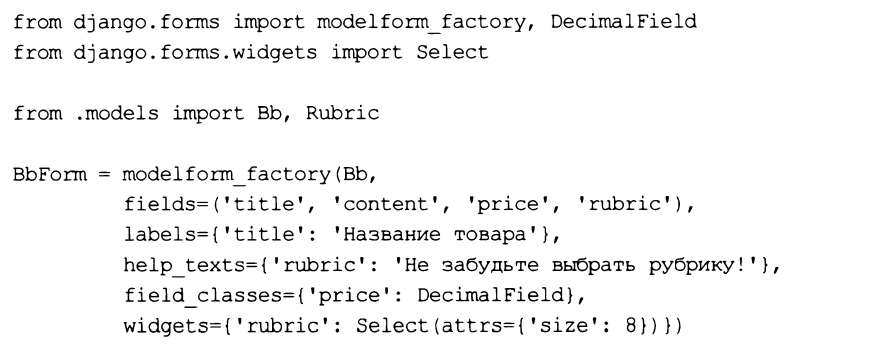
Параметр widgets позволяет задать элемент управления, которым будет представляться на веб-странице то или иное поле модели. Значением должен быть словарь, ключи элементов которого представляют имена полей формы, а значениями элементов станут экземпляры классов элементов управления или ссылки на сами эти классы.

Если какой-либо параметр не указан, то его значение либо будет взято из модели, либо установлено по умолчанию. Так, если не указать надпись для поля формы, то будет использовано название сущности, указанное в параметре verbose\_name конструктора поля модели, а если не указать тип элемента управления, то будет использован элемент управления по умолчанию для поля этого типа.

И наконец, параметр form СЛУЖИТ ДЛЯ указания базовой формы, связанной с моделью, на основе которой будет создана новая форма. Заданная форма может задавать какие-либо параметры, общие для целой группы форм.

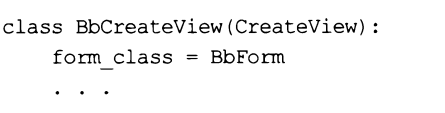
Функция modelform factory о в качестве результата возвращает готовый к использованию класс формы, связанной с моделью (подобные функции, генерирующие целые классы, называются фабриками классов).

Код, создающий на основе модели вь класс формы bbform с применением фабрики классов.

****

Ради эксперимента мы изменили надпись у поля названия товара, задали поясняющий текст у ПОЛЯ рубрики, сменили ТИП ПОЛЯ цены на decimaifieid и указали для поля рубрики представление в виде обычного списка высотой в 8 пунктов.

Класс, сохраненный в переменной bbform, можно использовать точно так же, как и любой написанный ’’вручную”, — например, указать его в контроллере-классе:



S

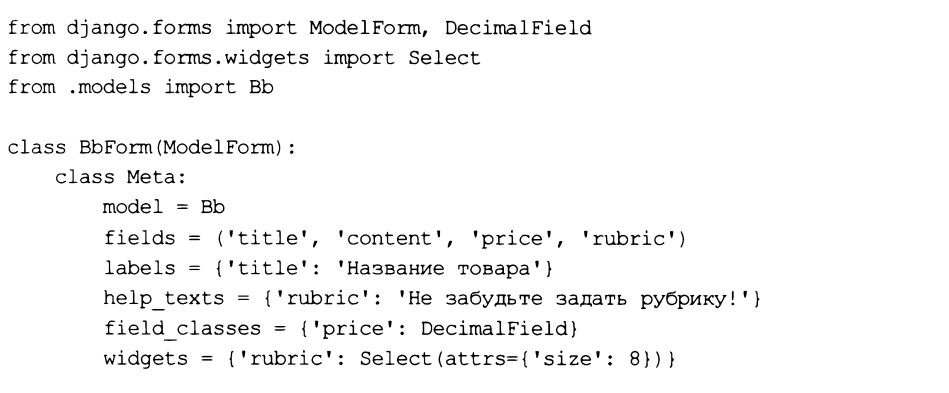
На основе фабрики классов удобно создавать в контроллерах-функциях редко используемые формы. В этом случае класс формы создается только при необходимости и уничтожается сразу же, как только перестанет существовать хранящая его переменная, благодаря чему экономится оперативная память.

Создание форм путем быстрого объявления если же форма, связанная с моделью, используется часто, то целесообразнее прибегнуть ко второму способу — объявить ее явно.

Класс формы, связанной с моделью, должен быть производным от класса modeiform из модуля django.forms. В этом классе объявляется вложенный класс Meta, в котором записывается набор атрибутов, имеющих те же имена, что параметры функции Modelform\_factory (), и то же назначение.

Такой способ объявления формы, при котором в ее классе записываются лишь общие указания, носит название быстрого объявления.

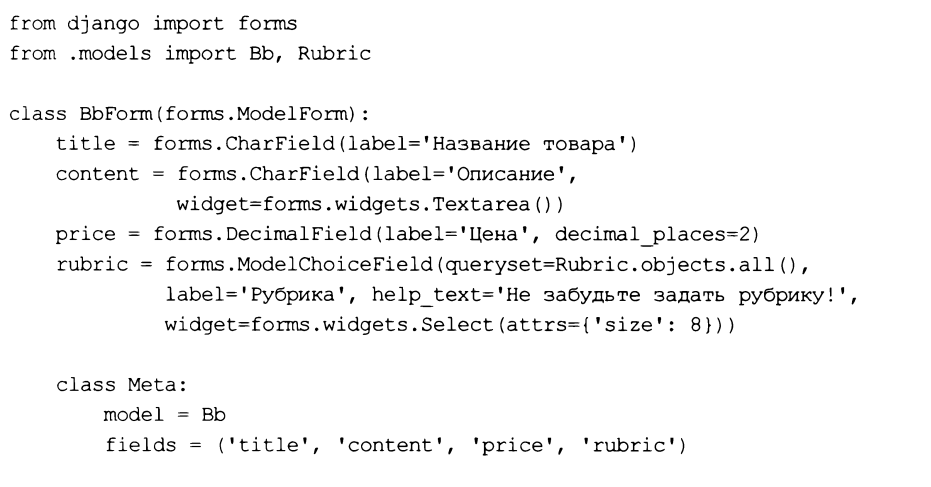
КОД класса формы bbform, созданный посредством быстрого объявления.

****

Создание форм путем полного объявления оба описанных ранее способа создания форм позволяли задать для ее полей весьма ограниченный набор параметров. Если же требуется описать поля формы во всех деталях, придется прибегнуть к третьему, сложному способу объявления.

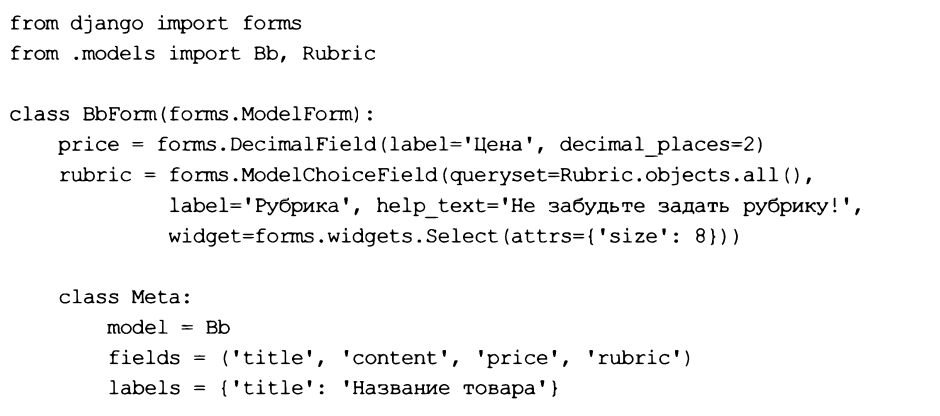
Как выполняется полное объявление при полном объявлении формы в ее классе детально описываются параметры как отдельных полей, так и — во вложенном классе Meta — самой формы. Полное объявление формы напоминает объявление модели.

Код полного объявления класса формы bbform, связанной с моделью вь.



При полном объявлении можно детально описать лишь некоторые поля формы, у которых требуется существенно изменить поведение. Параметры остальных полей формы можно указать путем быстрого объявления.

Там путем полного объявления создаются только поля price и rubric, а остальные поля созданы с применением быстрого объявления.

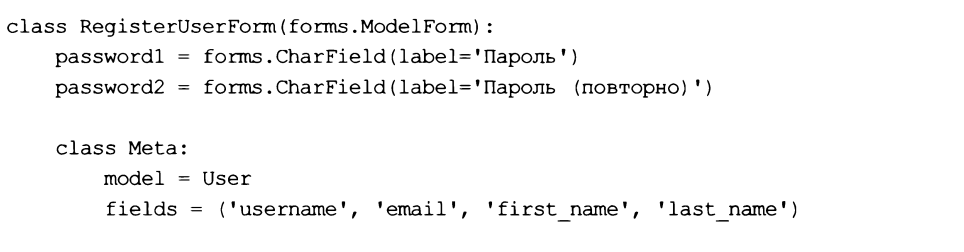
****

Применение полного объявления позволило, в частности, задать число знаков после запятой для поля типа decimaifieid (параметр decimal places). Быстрое объявление не даст это сделать.

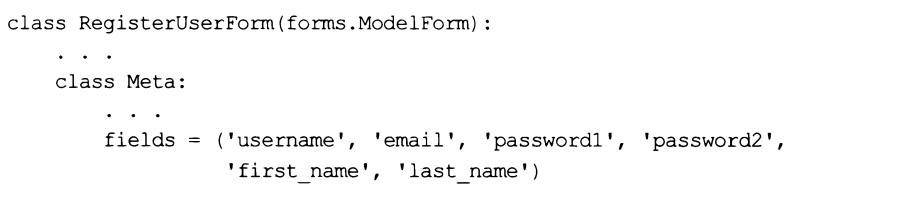
Внимание!

Если в классе формы присутствует и полное, и быстрое объявление какого-либо поля, то будет обработано только полное объявление. Параметры, записанные в быстром объявлении поля, будут проигнорированы.

Применяя полное объявление, можно добавить в форму дополнительные поля, отсутствующие в связанной с формой модели. Этот прием в коде, где В форму регистрации НОВОГО пользователя registeruserform добавлены поля Passwordl И password2, не существующие в модели User.

****

Поля, созданные таким образом, необязательно заносить в список из атрибута Fields вложенного класса Meta. Однако в этом случае такие поля при выводе формы на экран окажутся в ее конце. Чтобы задать для них нужное местоположение, их следует включить в список из атрибута fields по нужным позициям. Пример:

****

Параметры, поддерживаемые всеми типами полей поле, созданное путем полного объявления, представляется отдельным атрибутом класса формы. Ему присваивается экземпляр класса, представляющего поле определенного типа. Дополнительные параметры создаваемого поля указываются в соответствующих им именованных параметрах конструктора класса этого поля.

Рассмотрим параметры, поддерживаемые полями всех типов:

□ label — надпись для поля. Если не указан, то в качестве надписи будет использовано имя текущего поля;

□ help text— дополнительный поясняющий текст для текущего поля, который будет выведен возле элемента управления;

□ label suffix — суффикс, который будет добавлен к надписи для текущего поля.

Если параметр не указан, то будет взято значение одноименного параметра, поддерживаемого конструктором класса формы. Если и тот не указан, будет использовано значение по умолчанию — символ двоеточия;

□ initial — начальное значение для поля формы. Если не указан, то поле не будет иметь начального значения;

□ required— если True, то в поле обязательно должно быть занесено значение, если False, то поле может быть ’’пустым”. Значение по умолчанию — True;

□ widget — элемент управления, представляющий текущее поле на веб-странице.

Значение может представлять собой либо ссылку на класс элемента управления, либо экземпляр этого класса.

Если параметр не указан, будет использован элемент управления по умолчанию, применяемый для поля такого типа;

□ validators — валидаторы для текущего поля. Задаются в таком же формате, что и для поля модели;

□ error messages — сообщения об ошибках. Задаются в таком же формате, что и аналогичные сообщения для поля модели;

□ disabled— если True, то поле при выводе на экран станет недоступным, если False — доступным. Значение по умолчанию — False.

**Классы полей форм**

Все классы полей форм, поддерживаемые Django, объявлены в модуле Django.forms. Каждое такое поле предназначено для занесения значения строго определенного типа. Многие классы полей поддерживают дополнительные параметры, указываемые в вызовах конструкторов.

□ charfield —строковое или текстовое поле. Дополнительные параметры:

• min iength —минимальная длина значения, заносимого в поле, в символах;

• max length —максимальная длина значения, заносимого в поле, в символах;

• strip — если True, то из заносимого в поле значения будут удалены начальные и конечные пробелы, если False, то пробелы удаляться не будут (по умолчанию — True);

• empty vaiue— величина, которой будет представляться ’’пустое” поле (по умолчанию — ’’пустая” строка);

□ emailfieid— адрес электронной почты в строковом виде. Дополнительные параметры:

• min iength —минимальная длина почтового адреса в символах;

• max length —максимальная длина почтового адреса в символах;

□ urlfieid —интернет-адрес в виде строки. Дополнительные параметры:

• min iength —минимальная длина интернет-адреса в символах;

• max length —максимальная длина интернет-адреса в символах;

□ slugfield —слаг.

Поддерживается дополнительный параметр aiiow unicode. Если его значение равно True, то хранящийся в поле слаг может содержать символы Unicode, если False-- ТОЛЬКО СИМВОЛЫ ИЗ КОДИРОВКИ ASCII. Значение ПО умолчанию — False;

□ regexfieid— строковое значение, совпадающее с заданным регулярным выражением. Дополнительные параметры:

• regex — регулярное выражение. Может быть задано как в виде строки, так и в виде объекта типа ге;

• min iength —минимальная длина значения, заносимого в поле, в символах;

• max length —максимальная длина значения, заносимого в поле, в символах;

• strip — если True, то из заносимого в поле значения будут удалены начальные и конечные пробелы, если False, то пробелы удаляться не будут. Значение по умолчанию — False;

□ booieanfieid —логическое поле;

□ nullbooleanfield— ТО же Самое, ЧТО booieanfieid, НО дополнительно позволяет хранить значение null;

□ integerfieid— знаковое целочисленное поле обычной длины (32-разрядное).

Дополнительные параметры:

• min vaiue —минимальное значение, которое можно занести в поле;

• max value —максимальное значение, которое можно занести в поле;

□ floatfield —вещественное число. Дополнительные параметры:

• min vaiue —минимальное значение, которое можно занести в поле;

• max value —максимальное значение, которое можно занести в поле;

□ decimaifieid— вещественное число фиксированной точности, представленное объектом типа Decimal из модуля decimal Python. Дополнительные параметры:

• min vaiue —минимальное значение, которое можно занести в поле;

• max value —максимальное значение, которое можно занести в поле;

• max digits —максимальное количество цифр в числе;

• decimal places —количество цифр в дробной части числа;

□ datefieid — значение даты, представленное в виде объекта типа date из модуля Datetime Python.

Дополнительный параметр input formats задает последовательность поддерживаемых полем форматов даты. Значение по умолчанию определяется языковыми настройками проекта или параметром date input formats;

□ datetimefield— временная отметка В виде объекта типа datetime из модуля Datetime.

Дополнительный параметр input formats задает последовательность поддерживаемых полем форматов временных отметок. Значение по умолчанию определяется языковыми настройками проекта или параметром datetime input formats;

□ timefield— Значение времени В виде объекта типа time ИЗ модуля datetime Python.

Дополнительный параметр input formats задает последовательность поддерживаемых полем форматов времени. Значение по умолчанию определяется языковыми настройками проекта или параметром time input formats;

□ spiitdatetimefieid — то же самое, что и datetimefield, но для занесения значений даты и времени применяются разные элементы управления. Дополнительные параметры:

• input date formats — последовательность поддерживаемых полем форматов даты. Значение по умолчанию определяется языковыми настройками проекта или параметром date\_input\_formats;

• input time formats — последовательность поддерживаемых полем форматов

Времени. Значение по умолчанию определяется языковыми настройками проекта или параметром time\_input\_formats;

□ durationfield —промежуток времени, представленный объектом типа timedelta

Из модуля datetime Python;

□ modeichoicefieid— поле внешнего ключа вторичной модели, создающее связь "один-со-многими" или ”один-с-одним”. Позволяет выбрать в списке только одну связываемую запись первичной модели. Дополнительные параметры:

• queryset — набор записей первичной модели, на основе которого будет формироваться список;

• empty iabei — строка, обозначающая ’’пустой” пункт в списке связываемых записей. Значение по умолчанию: ”-------------”. Также можно убрать “пустой” пункт из списка, присвоив этому параметру значение None;

• to\_fieid name — имя поля первичной модели, значение из которого будет сохраняться в текущем поле внешнего ключа. Значение по умолчанию — None (обозначает ключевое поле);

□ modeimuitipiechoicefieid— поле внешнего ключа ведущей модели, создающее связь “ ногие-со-многими”. Позволяет выбрать в списке произвольное количество связываемых записей. Дополнительные параметры конструктора:

• queryset — набор записей ведомой модели, на основе которого будет формироваться список;

• to\_fieid name — имя поля ведомой модели, значение из которого будет сохраняться в текущем поле внешнего ключа. Значение по умолчанию — None (обозначает ключевое поле);

О choicefieid— поле со списком, в которое можно занести только те значения, xто приведены в списке. Значение записывается в поле в строковом формате.

Обязательный параметр choices задает перечень значений, которые будут представлены в списке, в виде:

• последовательности — в таком же формате, который применяется для задания параметра choices поля со списком моделей. Объекты перечисления не поддерживаются;

• ссылки на функцию, не принимающую параметров и возвращающую в качестве результата последовательность в таком же формате;

□ typedchoicefield — ТО же Самое, ЧТО choicefieid, НО позволяет хранить В Поле значение любого типа, а не только строкового. Дополнительные параметры:

• choices —перечень значений для выбора, в описанном ранее формате;

• coerce — ссылка на функцию, выполняющую преобразование типа значения, предназначенного для сохранения в поле. Должна принимать с единственным параметром исходное значение и возвращать в качестве результата то же значение, преобразованное к нужному типу;

• empty vaiue— значение, которым будет представляться ’’пустое” поле (по умолчанию — ’’пустая” строка).

Значение, представляющее ’’пустое” поле, можно записать непосредственно в последовательность, заданную в параметре choices;

□ multiplechoicefield— ТО же Самое, ЧТО choicefield, НО позволяет выбрать в списке произвольное число пунктов;

□ typedmultiplechoicefield— ТО же Самое, ЧТО И typedchoicefield, НО позволяет выбрать в списке произвольное число пунктов;

□ genericipaddressfieid— IP-адрес, записанный для протокола ipv4 или ipv6, в виде строки. Дополнительные параметры:

• protocol— допустимый протокол для записи IP-адресов, представленный в виде строки. Доступны значения: ’’ipv4”, "ipv6" и "both" (поддерживаются оба протокола). Значение по умолчанию: "both";

• inpack\_ipv4 — если True, то IP-адреса протокола ipv4, записанные в формате Ipv6, будут преобразованы к виду, применяемому в ipv4. Если False, то такое преобразование не выполняется. Значение по умолчанию— False. Этот параметр принимается во внимание, только если для параметра protocol указано значение "both".

□ uuidfieid — уникальный универсальный идентификатор, представленный объектом типа uuid из модуля uuid Python, в виде строки.

На заметку!

Также поддерживаются классы полей формы combofield И multivaluefield, ИЗ КОТОРЫХ первый применяется в крайне специфических случаях, а второй служит для разработки на его основе других классов полей. Описание этих двух классов можно найти на странице https://docs.djangoproject.eom/en/3.0/ref/forms/fields/.

Любому полю формы можно сопоставить элемент управления, посредством которого в него будет заноситься значение, указав его в параметре widget поля.

Классы элементов управления

Все классы элементов управления являются производными от класса widget из модуля django. Forms .widgets. Этот класс поддерживает параметр конструктора attrs, указывающий значения атрибутов HTML-тега, который помещается в код генерируемой страницы и создает элемент управления. Значением параметра должен быть словарь, ключи элементов которого совпадают с именами атрибутов тега, а значения элементов задают значения этих атрибутов.

Вот пример указания значения 8 для атрибута size тега <select>, создающего список (в результате будет создан обычный список высотой 8 пунктов):



Далее приведен список поддерживаемых Django классов элементов управления, которые также объявлены в модуле django. Forms .widgets:

□ Textinput —обычное поле ввода;

□ Numberinput —поле для ввода числа;

□ Emailinput —поле для ввода адреса электронной почты;

□ urlinput —поле для ввода интернет-адреса;

□ Passwordinput —поле для ввода пароля.

Поддерживается дополнительный параметр render value. Если присвоить ему значение True, то после неудачной валидации и повторного вывода формы на экран в поле ввода пароля будет подставлен набранный ранее пароль. Если задать параметру значение False, то поле будет выведено ’’пустым”. Значение по умолчанию — False;

□ Hiddeninput —скрытое поле;

□ Dateinput — поле для ввода значения даты.

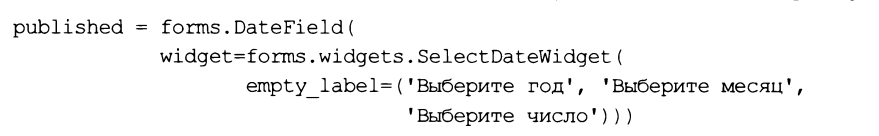
Дополнительный параметр format задает формат для представления значения даты. Если он не указан, будет использован формат, заданный языковыми настройками проекта, или первый формат из списка, хранящегося в параметре Date\_input\_FORMATS'.

□ selectdatewidget— ТО же, ЧТО И Dateinput, НО ВЫВОДИТ Три раскрывающихся списка: для выбора числа, месяца и года соответственно. Дополнительные параметры:

• years — список или кортеж значений года, которые будут выводиться в раскрывающемся списке, задающем год. Если не указан, будет выведен набор из текущего года и 9 следующих за ним годов;

• months — словарь месяцев, которые будут выводиться в раскрывающемся списке, задающем месяц. Ключами элементов этого словаря должны быть порядковые номера месяцев, начиная с 1 (1 — январь, 2— февраль и т. д.), а значениями элементов— названия месяцев. Если параметр не указан, то будут выведены все месяцы;

• empty iabei — величина, представляющая “пустое” значение числа, месяца и года. Может быть указана в виде строки (она будет использована для представления “пустых” даты, месяца и года), списка или кортежа из трех строковых элементов (первый будет представлять “пустой” год, второй — “пустой” месяц, третий — “пустое” число). Значение по умолчанию: “—”. Пример:

****

□ datetimeinput —поле для ввода временной отметки.

Дополнительный параметр format указывает формат выводимой временной отметки. Если не указан, то будет использовано значение, заданное языковыми настройками проекта, или первый формат из списка, хранящегося в параметре Datetime\_input\_formats

□ splitdatetimewidget —TO же, ЧТО И datetimeinput, НО ВВОД значений даты и времени осуществляется в разные поля. Дополнительные параметры:

• date format — формат выводимой даты. Если не указан, то будет взято значение, заданное языковыми настройками проекта, или первый формат из списка, хранящегося в параметре date input formats;

• time format — формат выводимого времени. Если не указан, то будет использовано значение, заданное языковыми настройками проекта, или первый Формат из списка, хранящегося в параметре time input formats;

• date attrs — значения атрибутов тега, создающего поля ввода даты;

• time attrs —значения атрибутов тега, создающего поля ввода времени.

Значения обоих параметров задаются в том же виде, что и у описанного ранее параметра attrs;

□ Timeinput — поле для ввода значения времени.

Дополнительный параметр format задает формат выводимого времени. Если не указан, то будет использовано значение, заданное языковыми настройками проекта, или первый формат из списка, хранящегося в параметре time input Formats;

□ Textarea —область редактирования;

□ Checkboxinput — флажок.

Дополнительному параметру check test можно присвоить ссылку на функцию, которая в качестве параметра принимает хранящееся в поле значение и возвращает True, если флажок должен быть выведен установленным, и False, если сброшенным;

□ Select — список, обычный или раскрывающийся (зависит от значения атрибута Size тега <seiect>), с возможностью выбора только одного пункта. Перечень выводимых пунктов он берет из параметра choices конструктора поля формы, с которым связан.

Перечень пунктов, выводимых в списке, также можно присвоить дополнительному параметру choices. Он задается в том же формате, что и у параметра Choices конструктора поля choicefieid, и имеет приоритет перед таковым, указанным в параметрах поля формы;

□ radioselect —аналогичен Select, но выводится в виде группы переключателей;

□ seiectmuitipie —то же самое, что и select, но позволяет выбрать произвольное число пунктов;

□ checkboxselectmultiple — аналогичен seiectmuitipie, НО ВЫВОДИТСЯ В виде набора флажков;

□ nuiibooieanseiect —раскрывающийся список с пунктами Да, Нет и Неизвестно.

1. Обработка форм

Высокоуровневые контроллеры-классы обработают и сохранят данные из формы самостоятельно. Но при использовании контроллеров-классов низкого уровня или контроллеров-функций обрабатывать формы придется вручную.

Создание формы для добавления записи для добавления записи следует создать экземпляр класса формы, поместить его в контекст шаблона и выполнить рендеринг шаблона, представляющего страницу добавления записи, тем самым выведя форму на экран. Все эти действия выполняются при отправке клиентом запроса с помощью HTTP-метода GET.

Экземпляр класса формы создается вызовом конструктора без параметров:



Если требуется поместить в форму какие-то изначальные данные, то используется необязательный параметр initial конструктора класса формы. Ему присваивается словарь, ключи элементов которого задают имена полей формы, а значения элементов — изначальные значения для этих полей. Пример:



Повторное создание формы после ввода посетителем данных в форму и нажатия кнопки отправки веб-обозреватель выполняет POST-запрос, отправляющий введенные в форму данные. Получив такой запрос, контроллер ’’поймет”, что нужно выполнить валидацию данных из формы и, если она пройдет успешно, сохранить эти данные в новой записи модели.

Сначала нужно создать экземпляр класса формы еще раз и поместить в него данные, полученные в составе POST-запроса. Это выполняется вызовом конструктора класса формы с передачей ему в качестве первого позиционного параметра словаря с полученными данными, который можно извлечь из атрибута post объекта запроса. Пример:

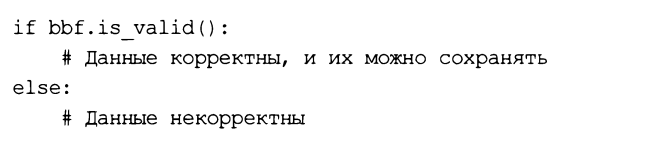
Bbf = bbform (request.POST)

После этого форма обработает полученные из запроса данные и подготовится к валидации.

Имеется возможность проверить, были ли в форму при ее создании помещены данные из запроса. Для этого достаточно вызвать метод isjoound (), поддерживаемый классом modelform. Метод вернет True, если при создании формы в нее были помещены данные из POST-запроса (т. Е. Выполнялось повторное создание формы), и False — в противном случае (т. Е. Форма создавалась впервые).

Валидация данных, занесенных в форму чтобы запустить валидацию формы, нужно выполнить одно из двух действий:

□ вызвать метод is vaiido формы. Он вернет True, если занесенные в форму данные корректны, и False —в противном случае. Пример:

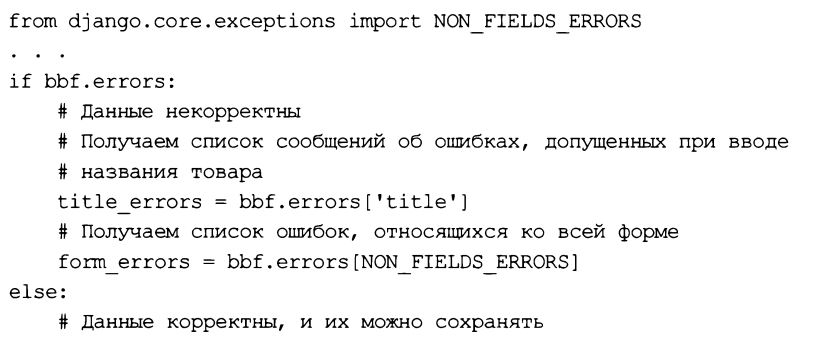
****

□ обратиться к атрибуту errors формы. Он хранит словарь с сообщениями о допущенных посетителем ошибках. Ключи элементов этого словаря совпадают с именами полей формы, а значениями являются списки текстовых сообщений об ошибках.

Ключ, совпадающий со значением переменной non field errors из модуля Django.core.exceptions, хранит сообщения об ошибках, относящихся не к определенному полю формы, а ко всей форме.

Если данные, занесенные в форму, корректны, то атрибут errors будет хранить “Пустой” словарь.

Пример:

****

Если данные корректны, то их следует сохранить и выполнить перенаправление на страницу со списком записей или содержанием только что добавленной записи, чтобы посетитель сразу смог увидеть, увенчалась ли его попытка успехом.

Если же данные некорректны, то необходимо повторно вывести страницу с формой на экран. В форме, рядом с элементами управления, будут показаны все относящиеся к ним сообщения об ошибках, и посетитель сразу поймет, что он сделал не так.

Сохранение данных, занесенных в форму сохранить данные, занесенные в связанную с моделью форму, можно вызовом метода save () формы:

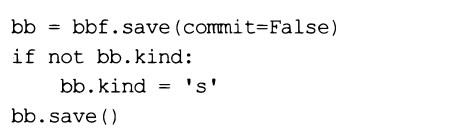


Перед сохранением данных из формы настоятельно рекомендуется выполнить их валидацию. Если этого не сделать, то метод save о перед сохранением выполнит валидацию самостоятельно и, если она не увенчалась успехом, возбудит исключение vaiueerror. А обрабатывать результат, возвращенный методом is vaiido, удобнее, чем исключение.

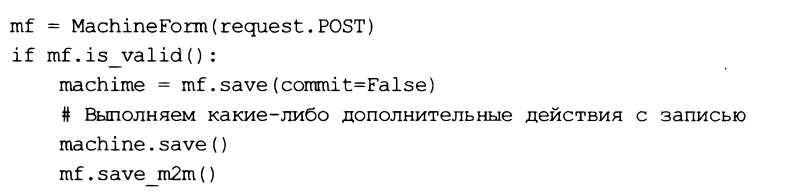
Метод save () в качестве результата возвращает объект созданной или исправленной записи модели, связанной с текущей формой.

Есть возможность получить только что созданную, но еще не сохраненную, запись модели, чтобы внести в нее какие-либо правки. Сделать это можно, записав в вызове метода save () необязательный параметр commit и присвоив ему значение False.

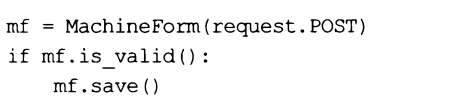
Объект записи будет возвращен методом save о в качестве результата, но сама запись сохранена не будет. Пример:

****

При сохранении записи модели, связанной с другой моделью связью "многие-со-многими”, нужно иметь в виду один момент. Чтобы связь между записями была успешно создана, связываемая запись должна иметь ключ (поскольку именно он записывается в связующей таблице). Однако пока запись не сохранена, ключа у нее нет. Поэтому сначала нужно сохранить запись вызовом метода save о у самой модели, а потом создать связь вызовом метода save\_m2m() формы. Вот пример:

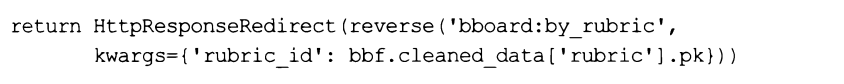
****

Отметим, что метод save\_m2m() нужно вызывать только в том случае, если сохранение записи ВЫПОЛНЯЛОСЬ ВЫЗОВОМ метода save () формы С параметром commit, равным False, и последующим вызовом метода save () модели. Если запись сохранялась вызовом save о формы без параметра commit (или если для этого параметра было указано значение по умолчанию True), то метод save\_m2m() вызывать не нужно — форма сама сохранит запись и создаст связь. Пример:

****

Доступ к данным, занесенным в форму, иногда бывает необходимо извлечь из формы занесенные в нее данные, чтобы, скажем, сформировать на их основе интернет-адрес перенаправления. Эти данные, приведенные к нужному типу (строковому, целочисленному, логическому и др.), хранятся в атрибуте cieaned\_data формы в виде словаря, ключи элементов которого совпадают с именами полей формы, а значениями элементов станут значения, введенные в эти поля.

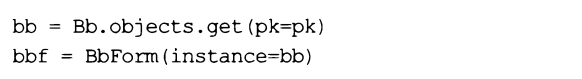
Вот пример использования ключа рубрики, указанной в только что созданном объявлении, для формирования адреса перенаправления:

****

Правка записи посредством формы чтобы исправить уже имеющуюся в модели запись посредством формы, связанной с моделью, нужно выполнить следующие шаги:

1. При получении запроса по HTTP-методу GET создать форму для правки записи.

В этом случае нужно указать исправляемую запись, задав ее в параметре Instance конструктора класса формы. Пример:

****

2. Вывести страницу с формой на экран.

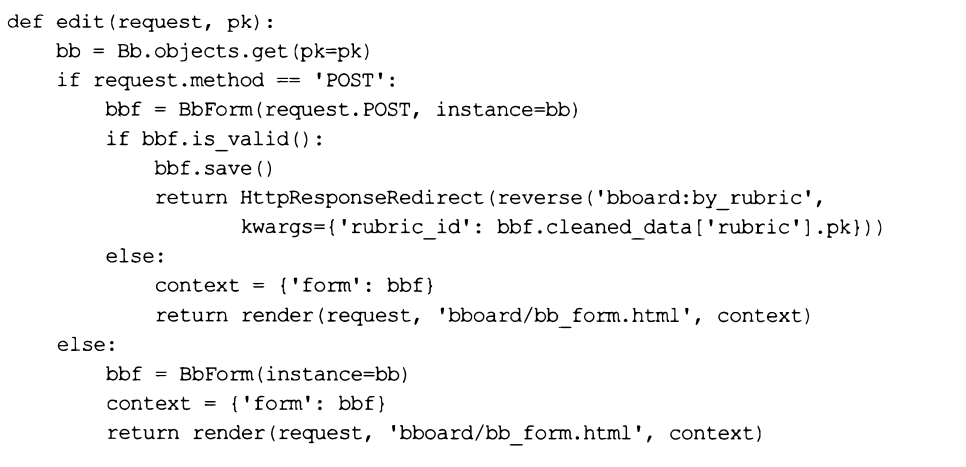
3. После получения POST-запроса с исправленными данными создать форму во второй раз, указав первым позиционным параметром полученные данные, извлеченные из атрибута post запроса, а параметром instance— исправляемую запись.

4. Выполнить валидацию формы:

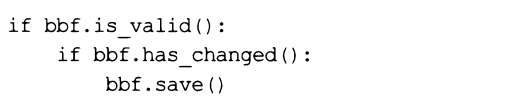
• если валидация прошла успешно, сохранить запись. После этого обычно выполняется перенаправление;

• если валидация завершилась неудачей, повторно вывести страницу с формой.

Приведен код контроллера-функции, осуществляющего правку записи, ключ которой был получен с URL-параметром pk. Здесь используется шаблон bboard\bb\_form.html.



При правке записи (и, в меньшей степени, при ее создании) может пригодиться метод has\_changed (). Он возвращает True, если данные в форме были изменены посетителем, и False —в противном случае. Пример:

****

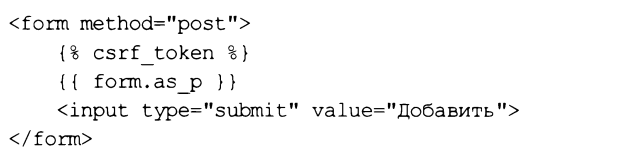
Атрибут changed data формы хранит список имен полей формы, значения которых были изменены посетителем.

1. Вывод форм на экран

**Быстрый вывод форм**

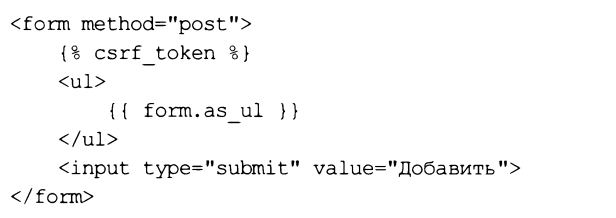
Быстрый вывод форм осуществляется вызовом одного метода из трех, поддерживаемых Django:

□ as\_p() — вывод по абзацам. Надпись для элемента управления и сам элемент управления, представляющий какое-либо поле формы, выводятся в отдельном абзаце и отделяются друг от друга пробелами. Пример использования:

****

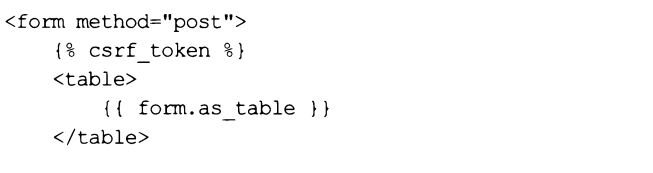
□ as ui () — вывод в виде маркированного списка. Надпись и элемент управления выводятся в отдельном пункте списка и отделяются друг от друга пробелами.

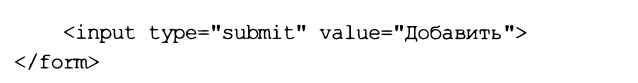
Теги <ui> и </ui> не формируются. Пример:

****

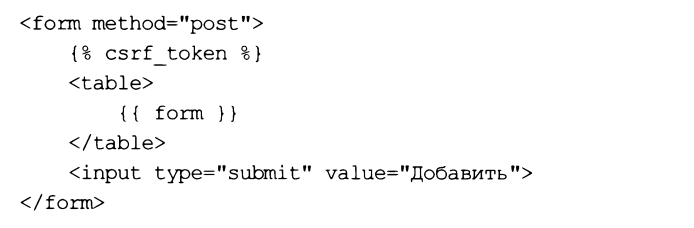
□ as table () — вывод в виде таблицы из двух столбцов: в левом выводятся надписи, в правом — элементы управления. Каждая пара “надпись—элемент управления” занимает отдельную строку таблицы. Теги <tabie> и </tabie> не выводятся.

Пример:





Можно просто указать переменную, хранящую форму, — метод as tableo будет вызван автоматически:

****

Обязательно уясним следующие моменты:

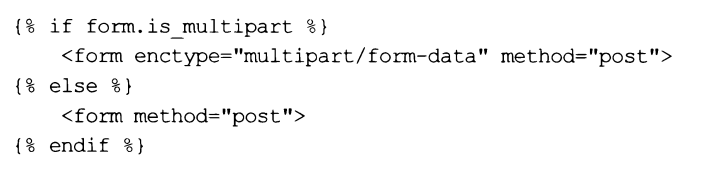
□ тег <form>, создающий саму форму, не выводится в любом случае. Его придется вставить в код шаблона самостоятельно;

□ кнопка отправки данных также не выводится, и ее тоже следует поместить в форму самостоятельно. Такая кнопка формируется одинарным тегом <input> с атрибутом type, значение которого равно "submit";

□ не забываем поместить в форму тег шаблонизатора csrf token. Он создаст в форме скрытое поле с электронным жетоном, по которому Django проверит, пришел ли запрос с того же самого сайта. Это сделано ради безопасности.

По умолчанию в веб-формах применяется метод кодирования данных Appiication/x-www-form-uriencoded. Но если форма отправляет файлы, то в ней нужно указать метод multipart/form-data.

Выяснить, какой метод следует указать в теге <form>, поможет метод Is muitipartо, поддерживаемый формой. Он возвращает True, если форма содержит поля, предназначенные для хранения файлов. И, соответственно, требует указания метода multipart/form-data, И False — в противном случае. Пример:



**Расширенный вывод форм**

Django предоставляет инструменты расширенного вывода форм, позволяющие располагать отдельные элементы управления произвольно.

Прежде всего экземпляр класса modeiform, представляющий связанную с моделью форму, поддерживает функциональность словаря. Ключи элементов этого словаря совпадают с именами полей формы, а значениями элементов являются экземпляры класса boundfieid, которые представляют отдельные поля формы.

Если указать в директиве шаблонизатора непосредственно экземпляр класса Boundfieid, то он будет выведен как HTML-код, создающий элемент управления для текущего поля. Вот так можно вывести область редактирования для ввода описания товара:



В результате мы получим такой HTML-код:



Еще класс boundfieid поддерживает следующие атрибуты:

□ label tag — HTML-код, создающий надпись для элемента управления, включая тег <1аье1>. Вот пример вывода кода, создающего надпись для области редактирования, в которую заносится описание товара:



Результирующий HTML-код:



□ label —только текст надписи;

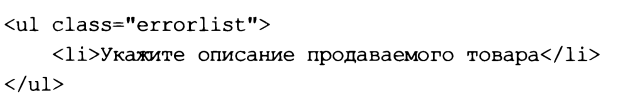
□ help text — дополнительный поясняющий текст;

□ errors —список сообщений об ошибках, относящихся к текущему полю.

Список ошибок можно вывести в шаблоне непосредственно:



В этом случае будет сформирован маркированный список с привязанным стилевым классом errorlist, а отдельные ошибки будут выведены как пункты этого списка, например:



Также можно перебрать список ошибок в цикле и вывести отдельные его элементы с применением любых других HTML-тегов.

Чтобы получить список сообщений об ошибках, относящихся ко всей форме, следует вызвать метод non field errors () формы:



□ is hidden — True, если это скрытое поле, False, если какой-либо иной элемент управления.

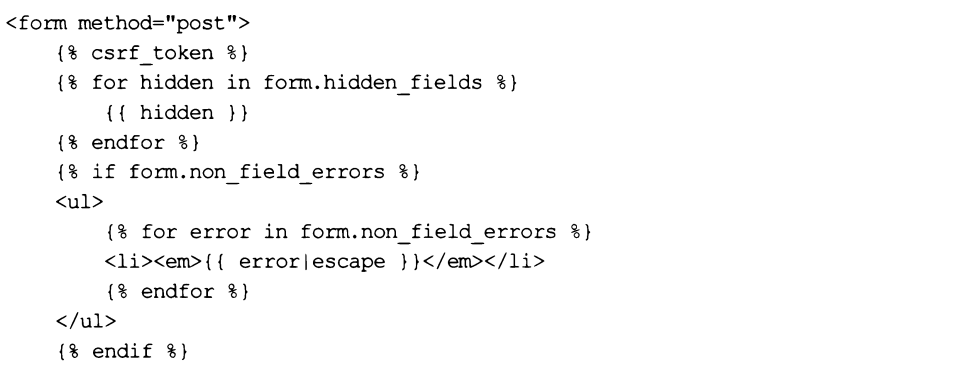
Методы класса modelform:

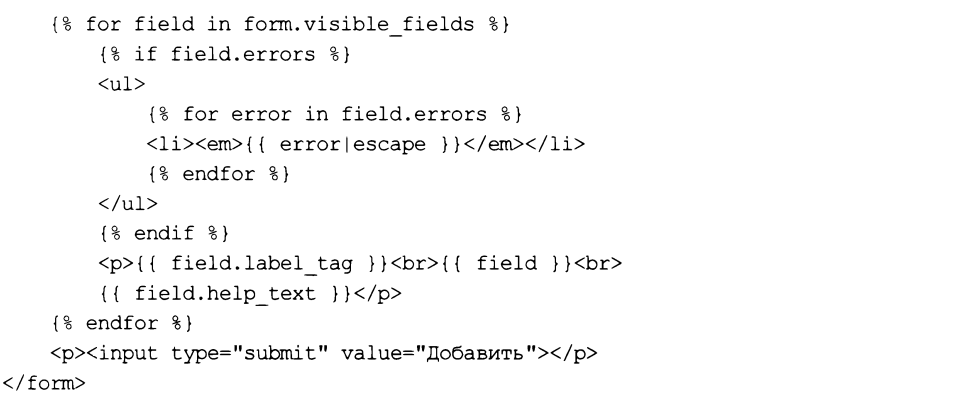
□ visibie fieids о — возвращает список видимых полей, которые представляются на экране обычными элементами управления;

□ hidden fieids о — возвращает список невидимых полей, представляющихся скрытыми полями HTML.

Приведен код, создающий форму для добавления объявления, в которой сообщения об ошибках выводятся курсивом; надписи, элементы управления и поясняющие тексты разделяются разрывом строки (HTML-тегом <br>); а невидимые поля (если таковые есть) выводятся отдельно от видимых. Сама форма показана на рис. 13.1.

Рис. 13.1. Веб-форма:

****

****

1. Валидация в формах

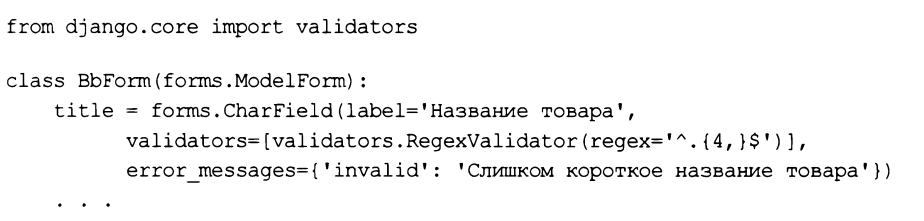
Валидацию можно выполнять не только в модели, но и в формах, связанных с моделями, применяя аналогичные инструменты.

**Валидация полей формы**

Валидацию отдельных полей формы можно реализовать двумя способами: с применением валидаторов или путем переопределения методов формы.

**Валидация с применением валидаторов**

Валидация с помощью валидаторов в полях формы выполняется так же, как и в полях модели. Вот пример проверки, содержит ли название товара больше четырех символов, с выводом собственного сообщения об ошибке:

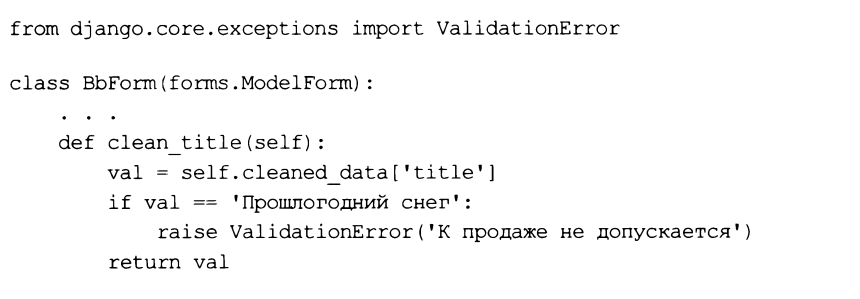


Разумеется, мы можем использовать не только стандартные валидаторы, объявленные в модуле django.core.validators, НО И СВОИ собственные.

**Валидация путем переопределения методов формы**

Более сложная валидация значения какого-либо поля реализуется в классе формы, в переопределенном методе с именем вида с1еап\_<имя поля>(self). Этот метод должен получать значение поля из словаря, хранящегося в атрибуте cieaned data, не должен принимать параметров и всегда обязан возвращать значение проверяемого поля. Если значение не проходит валидацию, то в методе следует возбудить исключение validationerror.

Вот пример проверки, не собирается ли посетитель выставить на продажу прошлогодний снег:

****

**Валидация формы**

Выполнить более сложную проверку или проверить значения сразу нескольких полей формы, т. Е. Провести валидацию формы, можно в переопределенном методе Clean (self) класса формы.

Метод не должен принимать параметров, возвращать результата, а обязан при неудачной валидации возбудить исключение validationerror, чтобы указать на возникшую ошибку. Первым же действием он должен вызвать одноименный метод базового класса, чтобы он заполнил словарь, хранящийся в атрибуте cieaned data (если мы этого не сделаем, то не сможем получить данные, занесенные в форму).

Далее приведен пример реализации такой же проверки, описание товара должно быть занесено, а значение цены должно быть неотрицательным.

