**Тема занятия № 19: Модуль 10. Контроллеры-классы**

1. Классы, выводящие одну запись

Контроллеры-классы ИЗ модуля django.views.generic.detail— более высокоуровневые, чем базовые. Они носят название обобщенных.

Поскольку выполняют типовые задачи и могут быть использованы в различных ситуациях.

Примесь singleobjectmixin: поиск записи класс-примесь singieobjectmixin, наследующий от contextmixin, выполняет сразу три действия:

□ извлекает из полученного интернет-адреса ключ или слаг записи, предназначенной к выводу на странице;

□ ищет запись в заданной модели по полученному ранее ключу или слагу;

□ помещает найденную запись в контекст шаблона.

Примесь поддерживает довольно много атрибутов и методов:

□ model — атрибут, задает модель;

□ queryset— атрибут, указывает либо диспетчер записей (Manager), либо набор записей (Queryset), в котором будет выполняться поиск записи;

□ get queryset (self) — метод, должен возвращать набор записей (queryset), в котором будет выполняться поиск записи.

В изначальной реализации возвращает значение атрибута queryset, если оно задано, или набор записей из модели, заданной в атрибуте model;

□ pk url kwarg — атрибут, задает имя URL-параметра, через который контроллер-класс получит ключ записи (по умолчанию: ”рк");

□ slug\_fieid— атрибут, задает имя поля модели, в котором хранится слаг (по умолчанию: "slug”);

□ get\_siug\_fieid(self) — метод, должен возвращать строку с именем поля модели, в котором хранится слаг. В реализации по умолчанию возвращает значение ИЗ атрибута slug\_field;

□ siug url kwarg — атрибут, задает имя URL-параметра, через который контроллер-класс получит слаг (по умолчанию: "slug");

□ query pk and slug — атрибут, указывает, что случится, если ключ записи не был найден в интернет-адресе. Если значение атрибута равно False, то в этом случае будет возбуждено исключение attributeerror, если True— будет выполнена попытка найти запись по слагу (если он присутствует — иначе также будет возбуждено исключение attributeerror);

□ context object name— атрибут, задает имя переменной контекста шаблона, в которой будет сохранена найденная запись;

□ get\_context\_object\_name(self, obj) — метод, должен возвращать имя переменной контекста шаблона, в которой будет сохранена найденная запись, в виде строки. В параметре obj передается объект записи.

В изначальной реализации возвращает значение атрибута context object name или, если этот атрибут хранит None, приведенное к нижнему регистру имя модели (так, если задана модель Rubric, метод вернет имя rubric);

□ get object (self, queryset=None) — метод, ВЫПОЛНЯЮЩИЙ ПОИСК записи ПО указанным критериям и возвращающий найденную запись в качестве результата.

В параметре queryset может быть передан набор записей, в котором должен выполняться поиск.

В изначальной реализации ищет запись в наборе из параметра queryset или, если он не задан, — в наборе записей, возвращенном методом get queryset ().

Значения ключа и слага получает из словаря, сохраненного в атрибуте экземпляра kwargs (его создает метод setup о), а имена необходимых URL-параметров— ИЗ атрибутов pk\_url\_kwarg И siug url kwarg. Если запись не найдена, метод возбуждает исключение Http404 из модуля django.http;

□ get\_context\_data(self, \*\*kwargs) — переопределенный метод, создающий и возвращающий контекст шаблона.

В изначальной реализации требует, чтобы в экземпляре текущего контроллера-класса присутствовал атрибут object, Хранящий найденную Запись ИЛИ None, если таковая не была найдена, а также если контроллер используется для создания новой записи. В контексте шаблона создает переменную object и переменную с именем, возвращенным методом get\_context\_object\_name(), обе переменные хранят найденную запись.

Примесь singleobjecttemplateresponsemixin:

Рендеринг шаблона на основе найденной записи класс-примесь singleobjecttemplateresponsemixin, наследующий ОТ templateresponsemixin, выполняет рендеринг шаблона на основе записи, найденной в модели. Он требует, чтобы в контроллере-классе присутствовал атрибут object, в котором хранится либо найденная запись в виде объекта модели, либо None, если запись не была найдена, а также если контроллер используется для создания новой записи.

Вот атрибуты и методы, поддерживаемые этим классом:

□ template name field— атрибут, содержащий ИМЯ ПОЛЯ модели, в котором хранится путь к шаблону. Если None, то путь к шаблону не будет извлекаться из записи модели (поведение по умолчанию);

□ template name suffix — атрибут, хранящий строку С суффиксом, который будет добавлен к автоматически сгенерированному пути к шаблону (по умолчанию: "\_detail”);

□ get\_tempiate\_names (self)— переопределенный метод, возвращающий список путей к шаблонам, заданных в виде строк.

В изначальной реализации возвращает список:

• либо из пути, извлеченного из унаследованного атрибута template name, если э тот путь указан;

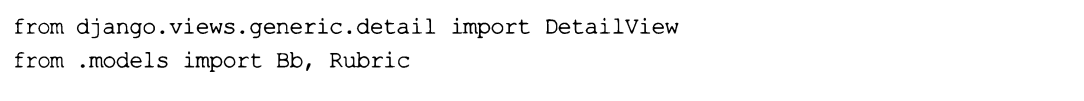
• либо из:

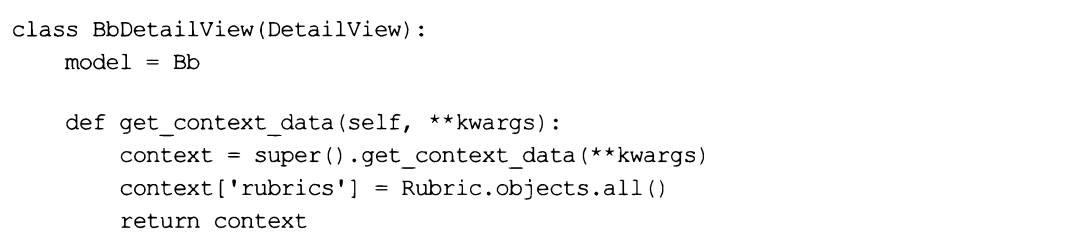
° пути, извлеченного из поля модели, имя которого хранится в атрибуте Template name field, если все необходимые данные (имя поля, запись модели и сам путь в поле этой записи) указаны;

П пути вида <псевдоним приложения>\<имя модели><суффикс>.Ыт\ (так, для модели вь из приложения bboard будет сформирован путь bboard\bb\_Detail.html).

Контроллер detailview: все вместе контроллер-класс Detailview наследует классы View, singleobj ectmixin И Singieobjecttempiateresponsemixin. Он ищет запись по полученным значениям ключа или слага, заносит ее в атрибут object (чтобы успешно работали наследуемые им примеси) и выводит на экран страницу с содержимым этой записи.

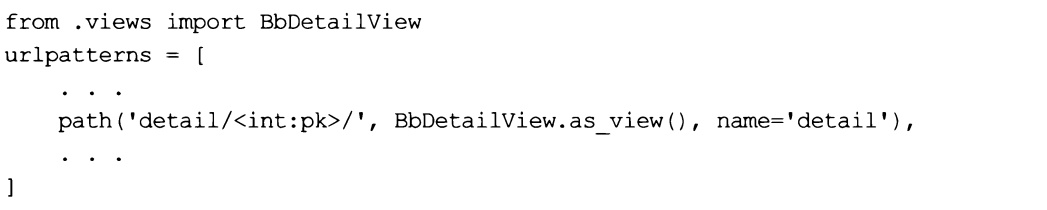
Приведен код контроллера-класса bbdetaiiview, производного от Detailview и выводящего страницу с объявлением, выбранным посетителем.





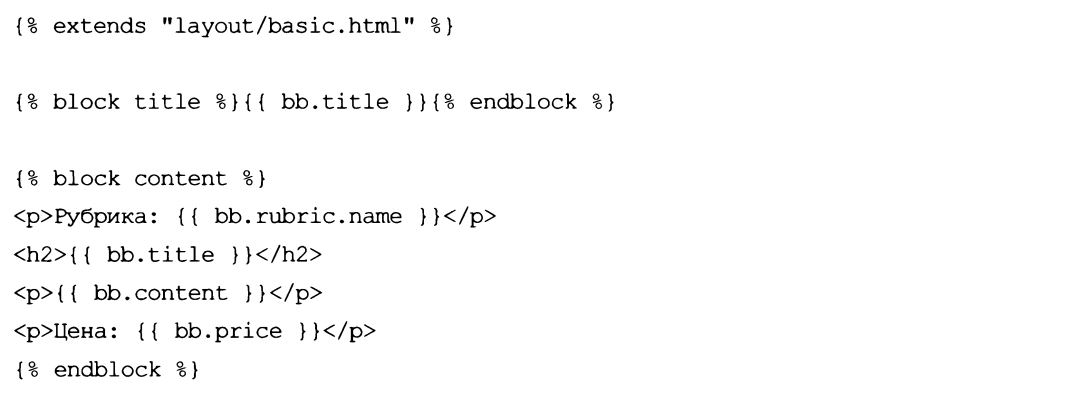
Компактность кода контроллера обусловлена в том числе и тем, что он следует соглашениям. Так, в нем не указан путь к шаблону — значит, класс будет искать шаблон со сформированным по умолчанию путем bboard\bb\_detail.html.

Добавим в список маршрутов маршрут, связанный с этим контроллером:

****

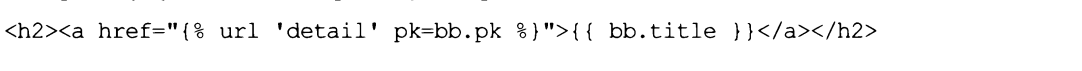
Опять же, этот маршрут написан соответственно соглашениям — у URL-параметра, содержащего ключ записи, указано имя pk, используемое классом Detailview по умолчанию.

Теперь напишем шаблон bboard\bb\_detail.html.

****

По умолчанию класс bbdetailview создаст в контексте шаблона переменную ьь, хранящую найденную запись (эту функциональность он унаследовал от базового класса Detailview). В коде шаблона мы используем эту переменную.

Осталось добавить в шаблоны bboard\index.html и bboard\by\_rubric.html гиперссылки, которые будут вести на страницу выбранного объявления:



Как видим, применяя контроллеры-классы достаточно высокого уровня и, главное, следуя заложенным в них соглашениям, можно выполнять весьма сложные действия без написания громоздкого кода.

1. Классы, выводящие наборы записей

Обобщенные классы из модуля django.views.generic.list выводят на экран целый набор записей.

**Примесь multipleobjectmixirr**

Извлечение набора записей класс-примесь muitipieobj ectmixin, наследующий от contextmixin, извлекает из модели набор записей, возможно, отфильтрованный, отсортированный и разбитый на части посредством пагинатора. Полученный набор записей он помещает в контекст шаблона.

Номер части, которую нужно извлечь, передается в составе интернет-адреса, через URL- или GET-параметр раде. Номер должен быть целочисленным и начинаться С 1. Если это правило нарушено, то будет возбуждено исключение Http404. Допустимо также указывать значение "last”, обозначающее последнюю часть.

Примесь поддерживает следующие атрибуты и методы:

□ model — атрибут, задает модель;

□ queryset — атрибут, указывает либо диспетчер записей (Manager), либо исходный набор записей (Queryset), из которого будут извлекаться записи;

□ get queryset (seif) — метод, должен возвращать исходный набор записей (Queryset), из которого будут извлекаться записи.

В изначальной реализации возвращает значение атрибута queryset, если оно задано, или набор записей из модели, которая задана в атрибуте model;

□ ordering — атрибут, задающий параметры сортировки записей. Значение может быть указано в виде:

• строки с именем поля — для сортировки только по этому полю. По умолчанию будет выполняться сортировка по возрастанию значения поля. Чтобы указать сортировку по убыванию, нужно предварить имя поля символом ’’Минус”;

• последовательности строк с именами полей — для сортировки сразу по нескольким полям.

Если значение атрибута не указано, станет выполняться сортировка, заданная в параметрах модели, или, когда таковое не указано, записи сортироваться не будут;

□ get ordering(seif) — метод, должен возвращать параметры сортировки записей. В изначальной реализации возвращает значение атрибута ordering;

□ paginate by— атрибут, задающий целочисленное количество записей в одной части пагинатора. Если не указан или его значение равно None, набор записей не будет разбиваться на части;

□ get paginate by(seif, queryset) — метод, должен возвращать число записей набора, полученного в параметре queryset, помещающихся в одной части пагинатора. В изначальной реализации просто возвращает значение из атрибута Paginate\_by;

□ page kwarg — атрибут, указывающий имя URL- или GET-параметра, через который будет передаваться номер выводимой части пагинатора, в виде строки (по умолчанию: ’’раде”);

□ paginate orphans— атрибут, задающий целочисленное минимальное число записей, которые могут присутствовать в последней части пагинатора. Если последняя часть пагинатора содержит меньше записей, то оставшиеся записи будут выведены в предыдущей части. Если задать значение 0, то в последней части может присутствовать сколько угодно записей (поведение по умолчанию);

□ get paginate orphans (self) — метод, должен возвращать минимальное число записей, помещающихся в последней части пагинатора. В изначальной реализации возвращает значение атрибута paginate orphans;

□ aiiow empty— атрибут. Значение True разрешает извлечение ’’пустой”, т.е. Не содержащей ни одной записи, части пагинатора (поведение по умолчанию). Значение False, напротив, предписывает при попытке извлечения ’’пустой” части возбуждать исключение Http404;

□ get aiiow empty(self) — метод, должен возвращать True, если разрешено извлечение ’’пустой” части пагинатора, или False, если такое недопустимо. В изначальной реализации возвращает значение атрибута aiiow empty;

□ paginator ciass — атрибут, указывающий класс используемого пагинатора (по умолчанию: Paginator ИЗ модуля django.core.paginator);

□ get paginator () — метод, должен создавать объект пагинатора и возвращать его в качестве результата. Формат объявления:

Get\_paginator(self, queryset, per\_page, orphans=0, Allоw\_empty\_first\_page=True) параметр queryset хранит обрабатываемый пагинатором набор записей, параметр per page — число записей в части, orphans — минимальное число записей в последней части пагинатора, a aiiow empty first page указывает, разрешено извлечение ’’пустой” части (True) или нет (False).

В изначальной реализации создает экземпляр класса пагинатора из атрибута Paginator ciass, передавая его конструктору все полученные им параметры;

□ paginate\_queryset (self, queryset, page\_size) —метод, разбивает набор записей, полученный в параметре queryset, на части с указанным в параметре page size числом записей в каждой части и возвращает кортеж из четырех элементов:

• объекта самого пагинатора;

• объекта его текущей части, номер которой был получен с URL- или getпараметром;

• набора записей, входящих в текущую часть (извлекается из атрибута Object list объекта текущей части пагинатора);

• True, если извлеченный набор записей действительно был разбит на части с применением пагинатора, и False — в противном случае;

□ context object name — атрибут, задает имя переменной контекста шаблона, в которой будет сохранен извлеченный набор записей;

О get\_context\_object\_name(seif, object\_iist) — метод, должен возвращать строку с именем переменной контекста шаблона, в которой будет сохранен набор записей, полученный в параметре object list.

В изначальной реализации возвращает имя из атрибута context object name или, если оно не указано, приведенное к нижнему регистру имя модели с добавленным суффиксом list;

□ get context data (seif, \*\*kwargs) — переопределенный метод, создающий и возвращающий контекст шаблона.

В изначальной реализации извлекает набор записей из необязательного параметра object list или, если этот параметр не указан, из атрибута object list.

После чего возвращает контекст с пятью переменными:

• object list— выводимый на странице набор записей (если используется пагинатор, это будет набор записей из его текущей части);

• переменная с именем, возвращенным методом get\_context\_object\_name(),— то же самое;

• is\_paginated— True, если применялся пагинатор, И False— В противном случае;

• paginator — объект пагинатора или None, если пагинатор не применялся;

• page obj — объект текущей страницы пагинатора или None, если пагинатор не применялся.

**Примесь multipleobjecttemplateresponsemixin:**

Рендеринг шаблона на основе набора записей класс-примесь multipleobjecttemplateresponsemixin, наследующий ОТ templateresponsemixin, выполняет рендеринг шаблона на основе извлеченного из модели набора записей. Он требует, чтобы в контроллере-классе присутствовал атрибут Object list, в котором хранится набор записей.

Вот список атрибутов и методов, поддерживаемых им:

□ template name suffix— атрибут, хранящий Строку С суффиксом, который будет добавлен к автоматически сгенерированному пути к шаблону (по умолчанию: "\_iist”);

О get tempiate names (self)— переопределенный метод, возвращающий список путей к шаблонам, заданных в виде строк.

В изначальной реализации возвращает список из:

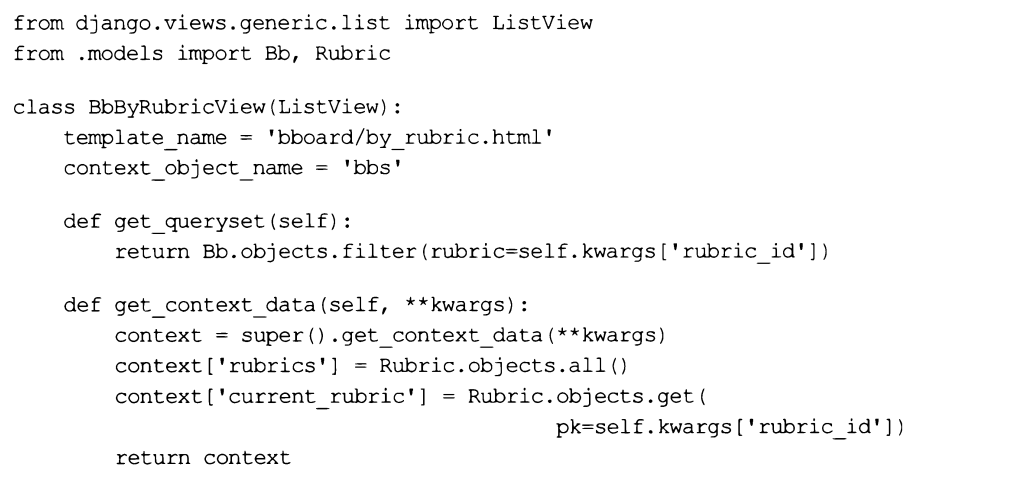
• пути, полученного из унаследованного атрибута template name, если этот путь указан;

• пути вида <псевдоним приложения>\<имя модели><суффикс>.Ыт\ (так, для модели вь из приложения bboard будет сформирован путь bboard\bb\_list.html).

**Контроллер listview**

Все вместе контроллер-класс Listview наследует классы View, muitipieobj ectmixin и Muitipieobj ecttempiateresponsemixin. Он извлекает из модели набор записей, записывает его в атрибут object list (чтобы успешно работали наследуемые им примеси) и выводит на экран страницу со списком записей.

Приведен код контроллера-класса bbbyrubricview, унаследованного от Listview и выводящего страницу с объявлениями из выбранной посетителем рубрики.

****

Код этого контроллера получился более объемным, чем у ранее написанного контроллера-функции by rubrico. Это обусловлено особенностями используемого нами шаблона bboard\by\_rubric.html. Во-первых, требуется вывести список рубрик и текущую рубрику, следовательно, придется добавить все эти данные в контекст шаблона, переопределив метод get\_context\_data ().

Во-вторых, мы используем уже имеющийся шаблон, поэтому вынуждены указать в контроллере его имя и имя переменной контекста, в которой будет храниться список объявлений.

Значение URL-параметра rubric id мы получили обращением к словарю из атрибута kwargs, содержащему все URL-параметры, указанные в маршруте. Из контекста шаблона извлечь его мы не можем.

1. Классы, работающие с формами

Обобщенные контроллеры-классы из модуля django.views.generic.edit рассчитаны на работу с формами, как связанными с моделями, так и обычными.

**Классы для вывода и валидации форм**

Классы самого низкого уровня ’’умеют” лишь вывести форму, проверить занесенные в нее данные на корректность и, в случае ошибки, вывести повторно, вместе с предупреждающими сообщениями.

**Примесь formmixin: создание формы**

Класс-примесь formmixin, производный от класса contextmixin, создает форму (неважно, связанную с моделью или обычную), проверяет введенные в нее данные, выполняет перенаправление, если данные прошли проверку, или выводит форму повторно (в противном случае).

Вот набор поддерживаемых атрибутов и методов:

□ form class — атрибут, хранит ссылку на класс используемой формы;

□ get form ciass (self) — метод, должен возвращать ссылку на класс используемой формы. В изначальной реализации возвращает значение атрибута form Class;

□ initial — атрибут, хранящий словарь с изначальными данными для занесения в только что созданную форму. Ключи элементов этого словаря должны соответствовать полям формы, а значения элементов зададут значения полей. По умолчанию хранит пустой словарь;

□ get initial (self) — метод, должен возвращать словарь с изначальными данными для занесения в только что созданную форму. В изначальной реализации просто возвращает значение атрибута initial;

□ success uri — атрибут, хранит интернет-адрес для перенаправления, если введенные в форму данные прошли проверку на корректность;

□ get success uri (self) — метод, должен возвращать интернет-адрес для перенаправления в случае, если введенные в форму данные прошли валидацию. В изначальной реализации возвращает значение атрибута success\_uri;

П prefix — атрибут, задает строковый префикс для имени формы, который будет присутствовать в создающем форму HTML-коде. Префикс стоит задавать только в том случае, если планируется поместить несколько однотипных форм в одном Теге <form>. По умолчанию хранит значение None (отсутствие префикса);

□ get\_prefix(self) — метод, должен возвращать префикс для имени формы.

В изначальной реализации возвращает значение из атрибута prefix;

П get\_form(self, form\_class=None) — метод, создающий И возвращающий объект формы.

В изначальной реализации, если класс формы указан в параметре form ciass, создает экземпляр этого класса, в противном случае — экземпляр класса, возвращенного методом get\_form\_class (). При этом конструктору класса формы передаются параметры, возвращенные методом get form kwargs ();

□ get form kwargs (self) — метод, должен создавать и возвращать словарь с параметрами, которые будут переданы конструктору класса формы в методе Get—form ().

В изначальной реализации возвращает словарь с элементами:

• initial — словарь с изначальными данными, возвращенный методом get Initial();

• prefix — префикс ДЛЯ имени формы, возвращенный методом get prefix ();

Следующие два элемента создаются только в том случае, если для отправки запроса применялись HTTP-методы POST и PUT (т. Е. При проверке введенных в форму данных):

• data — словарь с данными, занесенными в форму посетителем;

• files — словарь с файлами, отправленными посетителем из формы;

□ get—context—data (seif, \*\*kwargs) — переопределенный метод, создающий и возвращающий контекст шаблона.

В изначальной реализации добавляет в контекст шаблона переменную form, хранящую созданную форму;

□ form\_valid (self, form) — метод, должен выполнять обработку данных, введенных в переданную через параметр form форму, в том случае, если они прошли валидацию.

В изначальной реализации просто выполняет перенаправление по адресу, возвращенному методом get-success-url ();

□ form\_invalid (seif, form) — метод, должен выполнять обработку ситуации, когда данные, введенные в переданную через параметр form форму, не проходят валидацию. В изначальной реализации повторно выводит страницу с формой на экран.

**Контроллер processformview:**

Вывод и обработка формы

Контроллер-класс processformview, производный от класса view, выводит на экран страницу с формой, принимает введенные данные и проводит их валидацию.

Он переопределяет три метода, унаследованные от базового класса:

□ get (self, request, \*args, \*\*kwargs) — ВЫВОДИТ Страницу С формой на экран;

□ post (self, request, \*args, \*\*kwargs) — получает введенные в форму данные и выполняет их валидацию. Если валидация прошла успешно, вызывает метод Form\_valid (), В ПРОТИВНОМ Случае — метод form\_invalid ();

О put (self, request, \*args, \*\*kwargs)—TO же, ЧТО И post ().

**Контроллер-класс formview:**

Создание, вывод и обработка формы

Контроллер-класс formview, производный ОТ formmixin, processformview и Tempiateresponsemixin, создает форму, выводит на экран страницу с этой формой, проверяет на корректность введенные данные и, в случае отрицательного результата проверки, выводит страницу с формой повторно. Нам остается только реализовать обработку корректных данных, переопределив метод form valid ().

Приведен код контроллера-класса bbaddview, добавляющего на виртуальную доску новое объявление.



При написании этого класса мы столкнемся с проблемой. Чтобы после сохранения объявления сформировать интернет-адрес для перенаправления, нам нужно получить значение ключа рубрики, к которой относится добавленное объявление.

Поэтому мы переопределили метод get\_form(), в котором сохранили созданную форму в атрибуте object. После этого в коде метода get\_success\_url () без проблем сможем получить доступ к форме и занесенным в нее данным.

Сохранение введенных в форму данных мы выполняем в переопределенном методе Form\_valid().

**Классы для добавления, правки и** **удаления записей**

Описанные далее высокоуровневые классы, помимо обработки форм, выполняют добавление, правку и удаление записей.

Примесь modelformmixin: создание формы, связанной с моделью класс-примесь modelformmixin, наследующий ОТ классов singleobj ectmixin и Formmixin, полностью аналогичен последнему, но работает с формами, связанными с моделями.

Вот список поддерживаемых им атрибутов и методов:

□ model — атрибут, задает ссылку на класс модели, на основе которой будет создана форма;

□ fields — атрибут, указывает последовательность имен полей модели, которые должны присутствовать в форме.

Можно указать либо модель и список ее полей в атрибутах model и fields, либо Непосредственно класс формы в атрибуте form class, унаследованном от класса Formmixin, но никак не одновременно и то, и другое.

Если указан атрибут model, обязательно следует задать также и атрибут fields. Если этого не сделать, возникнет ошибка;

□ get form ciass (self) — переопределенный метод, должен возвращать ссылку на класс используемой формы.

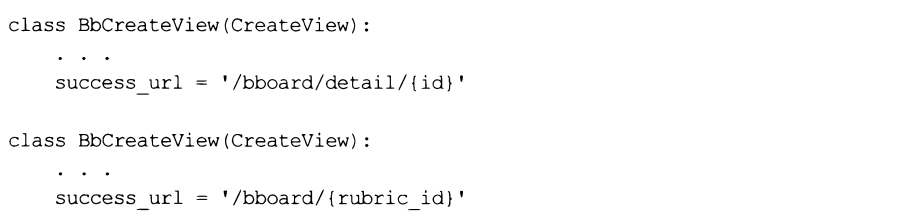
В изначальной реализации возвращает значение атрибута form class, если оно задано. В противном случае возвращается ссылка на класс формы, автоматически созданный на основе модели, которая взята из атрибута model или извлечена из набора записей, заданного в унаследованном атрибуте queryset. Для создания класса формы также используется список полей из атрибута fields;

□ success uri — атрибут, хранит интернет-адрес для перенаправления, если введенные в форму данные прошли проверку на корректность.

В отличие от одноименного атрибута базового класса formmixin, он поддерживает указание непосредственно в строке с интернет-адресом специальных последовательностей символов вида {<имя поля таблицы в базе данных>}. Вместо такой последовательности будет подставлено значение поля с указанным именем.

Отметим, что в такие последовательности должно подставляться имя не поля модели, а таблицы базы данных, которая обрабатывается моделью. Так, для вставки в адрес ключа записи следует использовать поле id, а не pk, а для вставки внешнего ключа — поле rubric id, а не rubric.

Примеры:



□ get success url (seif) — переопределенный метод, возвращает адрес для перенаправления в случае, если введенные данные прошли валидацию.

В изначальной реализации возвращает значение атрибута success uri, в котором последовательности вида {<имя поля таблицы в базе данных>} уже заменены значениями соответствующих полей. Если адрес там не указан, пытается получить его вызовом метода get\_absoiute\_uri () модели;

□ get form kwargs (self) — переопределенный метод, создает и возвращает словарь с параметрами, которые будут переданы конструктору класса формы в унаследованном методе get form ().

В изначальной реализации добавляет в словарь, сформированный унаследованным методом, элемент instance, хранящий обрабатываемую формой запись модели (если она существует, т.е. Форма используется не для добавления записи).

Эта запись извлекается из атрибута object;

□ form\_vaiid(self, form) — переопределенный метод, должен выполнять обработку данных, введенных в переданную через параметр form форму, в том случае, если они прошли валидацию.

В изначальной реализации сохраняет содержимое формы в модели, вызвав у нее метод save о, присваивает новую запись атрибуту object, после чего вызывает унаследованный Метод form valid ().

**Контроллер createview: создание новой записи**

Контроллер-класс createview наследует ОТ классов processformview, modelformmixin и singleobjecttemplateresponsemixin. Он ВЫВОДИТ форму, проверяет введенные в нее данные и создает на их основе новую запись.

Атрибут template name suffix этого класса хранит строку с суффиксом, который удет добавлен к автоматически сгенерированному пути к шаблону (по умолчанию: " form”).

Также в классе доступен атрибут object, в котором хранится созданная в модели запись или None, если таковая еще не была создана.

Пример контроллера-класса, производного от createview.

**Контроллер updateview: исправление записи**

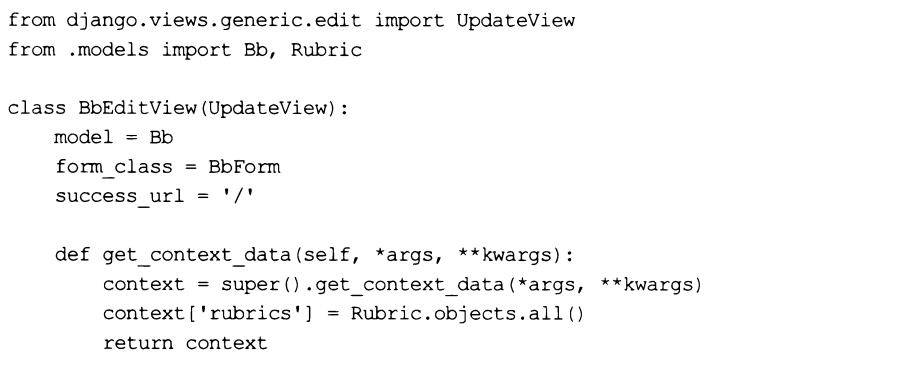
Контроллер-класс updateview наследует ОТ классов processformview, modelformmixin и singleobjecttemplateresponsemixin. Он ищет запись ПО полученным ИЗ urlпараметра ключу или слагу, выводит страницу с формой для ее правки, проверяет и сохраняет исправленные данные.

Атрибут tempiate name suffix этого класса хранит строку с суффиксом, который будет добавлен к автоматически сгенерированному пути к шаблону (по умолчанию: "form”).

Также в классе доступен атрибут object, в котором хранится исправляемая запись.

Поскольку класс updateview предварительно выполняет поиск записи, в нем необходимо указать модель (в унаследованном атрибуте model), набор записей (в атрибуте queryset) ИЛИ переопределить метод get\_queryset ().

Приведен код контроллера-класса bbeditview, который выполняет исправление объявления.

****

Шаблон bboard\bb\_form.html вы можете написать самостоятельно, взяв за основу уже имеющийся шаблон bboard\create.html (там всего лишь придется поменять текст добавление на Исправление, а подпись кнопки— с Добавить на Сохранить).

Также самостоятельно вы можете записать маршрут для этого контроллера и вставить в шаблоны bboard\index.html и bboard\by\_rubric.html код, создающий гиперссылки на страницы исправления объявлений.

**Примесь deletionmixin: удаление записи**

Класс-примесь deletionmixin добавляет наследующему ее контроллеру инструменты для удаления записи. Он предполагает, что запись, подлежащая удалению, уже найдена и сохранена в атрибуте object.

Класс объявляет атрибут success\_uri и метод get\_success\_uri (), аналогичные присутствующим в примеси modelformmixin.

**Контроллер deletevfew:**

Удаление записи с подтверждением

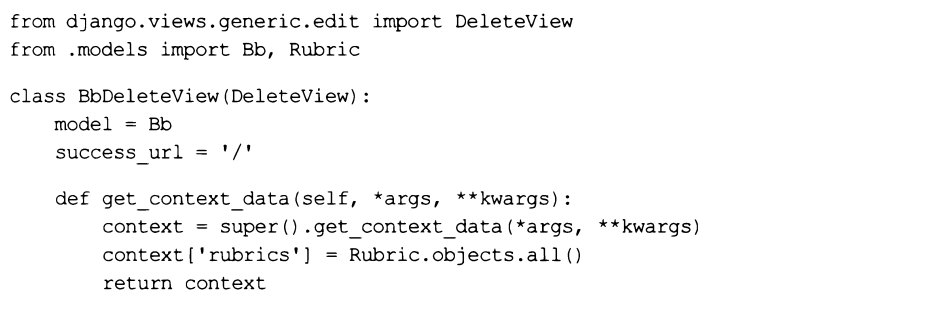
Контроллер-класс deleteview наследует ОТ классов Detailview, deletionmixin и Singieobjecttempiateresponsemixin. Он ищет запись по полученному из URL-napaметра ключу или слагу, выводит страницу подтверждения, включающую в себя форму с кнопкой удаления, и удаляет запись.

Атрибут template\_name\_suffix ЭТОГО Класса хранит строку С суффиксом, который будет добавлен к автоматически сгенерированному пути к шаблону (по умолчанию: "\_confirm\_delete”).

Также в классе доступен атрибут object, в котором хранится удаляемая запись.

Поскольку класс Deleteview предварительно выполняет поиск записи, в нем необходимо указать модель (в атрибуте model), набор записей (в атрибуте queryset) или переопределить метод get\_queryset ().

Приведен код контроллера-класса bbdeieteview, который выполняет удаление объявления.



Код шаблона bboard\bb\_confirm\_delete.html, выводящего страницу подтверждения и форму с кнопкой Удалить, приведен.

