**Тема занятия № 12: Контроллеры. Классы и функции**

1. Специальные ответы

Иногда нужно отправить посетителю данные формата, отличного от веб-страницы или простого текста. Для таких случаев Django предлагает три класса специальных ответов, объявленные в модуле django.http.

**Потоковый ответ**

Обычный ответ httpresponse полностью формируется в оперативной памяти. Если объем ответа невелик, это вполне допустимо. Но для отправки страниц большого объема этот класс не годится, поскольку отнимет много памяти. В таких случаях применяется потоковый ответ, который формируется и отсылается по частям небольших размеров.

Потоковый ответ представляется классом streaminghttpresponse. Формат его конструктора:

Streaminghttpresponse(<содержимое>[, content\_type=None][, status=200][,Reason=None])

Содержимое задается в виде последовательности строк или итератора, на каждом проходе возвращающего строку. Остальные параметры такие же, как и у класса Httpresponse.

Класс поддерживает следующие атрибуты:

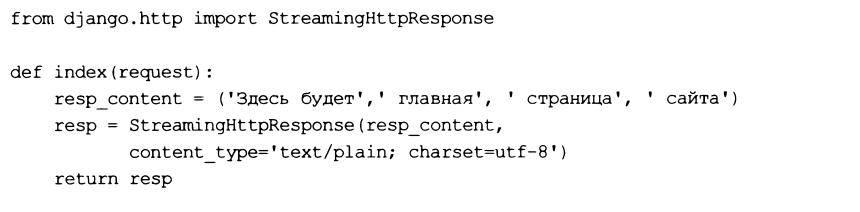
□ streaming content — итератор, на каждом проходе возвращающий фрагмент содержимого ответа в виде объекта типа bytes;

О status code — целочисленный код статуса;

□ reason phrase — СТРОКОВЫЙ статус;

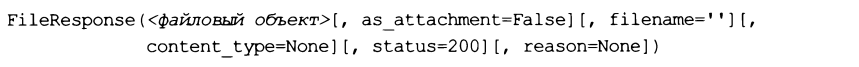
□ streaming — если True, ЭТО ПОТОКОВЫЙ Ответ, если False — обычный. Будучи вызванным у экземпляра класса streaminghttpresponse, метод всегда возвращает True.

Пример кода, выполняющего отправку потокового ответа, приведен.

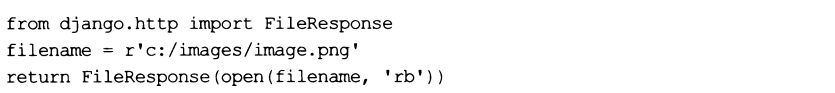
****

**Отправка файлов**

Для отправки клиентам файлов применяется класс fileresponse — производный от класса streaminghttpresponse:



Пример:

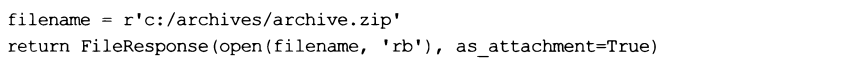
****

Отправленный таким образом файл будет открыт непосредственно в веб-обозревателе. Чтобы дать веб-обозревателю указание сохранить файл на локальном диске, достаточно задать в вызове конструктора класса fileresponse параметры:

□ as\_attachment СО значением True;

□ filename, в котором указать имя сохраняемого файла, — если заданный первым параметром файловый объект не содержит имени файла (например, если он был сформирован программно в оперативной памяти).

Пример:

****

**Отправка данных в формате JSON**

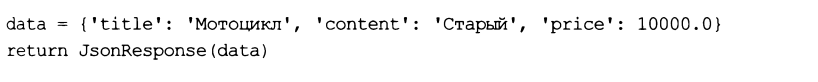
Для отправки данных в формате JSON применяется класс jsonresponse— производный от класса httpresponse. Формат его конструктора:



Кодируемые в JSON данные должны быть представлены в виде словаря Python.

Если требуется закодировать и отправить что-либо отличное от словаря, нужно назначить параметру safe значение False. Параметр encoder задает кодировщик, применяемый для преобразования данных в формат JSON; если он не указан, используется стандартный кодировщик.

Пример:

****

1. Сокращения Django

Сокращение — это функция, выполняющая сразу несколько действий. Применение сокращений позволяет несколько уменьшить код и упростить программирование.

Все сокращения, доступные в Django, объявлены в модуле django.shortcuts:

О render(<запрос>, <путь к шаблону>[, context=None][, content\_type=None][,Status=200]) — выполняет рендеринг шаблона и отправку получившейся в результате страницы клиенту, запрос должен быть представлен в виде экземпляра класса Request, путь к шаблону— В виде строки.

Необязательный параметр context указывает контекст шаблона, content type — MIME-тип и кодировку отправляемого ответа (по умолчанию text/html с кодировкой из параметра default charset настроек проекта), a status — числовой код статуса (по умолчанию — 2 о о).

В качестве результата возвращается готовый ответ в виде экземпляра класса Httpresponse.

Пример:



О redirect (<цель>[, permanent=False] [, <значения URL-параметров:?] ) — выполняет перенаправление по заданной цели, в качестве которой могут быть указаны:

• объект модели — тогда интернет-адрес для перенаправления будет получен вызовом метода get\_absoiute\_uri () этого объекта ;

• имя маршрута (возможно, с указанием пространства имен) и набор значении Url-параметров — тогда адрес для перенаправления будет сформирован с применением обратного разрешения.

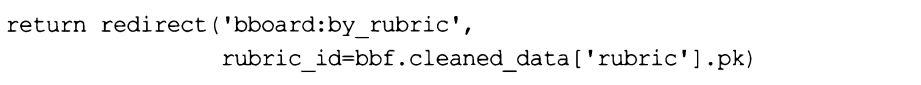
Значения url-параметров могут быть указаны в вызове функции в виде как позиционных, так и именованных параметров;

• непосредственно заданный интернет-адрес.

Необязательный параметр permanent указывает тип перенаправления: временное (если False или если он опущен) или постоянное (если True).

В качестве результата возвращается полностью сформированный экземпляр класса httpresponseredirect.

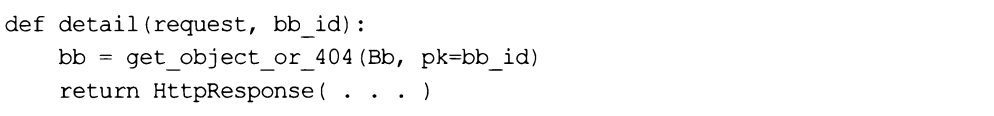
Пример:

****

□ get\_object\_or\_404 (<источник>, <условия поиска>) — ищет запись согласно заданным условиям поиска и возвращает ее в качестве результата. Если запись найти не удается, возбуждает исключение Http404. В качестве источника можно указать класс модели, диспетчер записей (экземпляр класса Manager) или набор записей (экземпляр класса queryset).

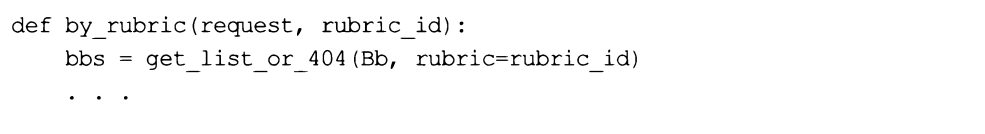
Если заданным условиям поиска удовлетворяют несколько записей, то возбуждается исключение multipleobjectsreturned.

Пример:

****

О get\_iist\_or\_404 (<источник>, <условия фильтрации>) — применяет к записям заданные условия фильтрации и возвращает в качестве результата полученный набор записей (экземпляр класса Queryset). Если ни одной записи, удовлетворяющей условиям, не существует, то возбуждает исключение Http404. В качестве источника можно указать класс модели, диспетчер записей или набор записей.

Пример:

****

1. Программное разрешение интернет-адресов

Иногда может понадобиться программно ’’прогнать” какой-либо интернет-адрес через маршрутизатор, выяснить, совпадает ли он с каким-либо маршрутом, и получить сведения об этом маршруте, т. Е. Выполнить программное разрешение адреса.

Для этого предназначена функция resolve() из модуля django.urls:



Интернет-адрес указывается в виде строки.

Параметр uriconf задает путь к модулю со списком маршрутов, который будет использоваться для программного разрешения. Если он не указан, задействуется модуль со списком маршрутов уровня проекта, заданный в его настройках (параметр root urlconf. Более подробно о нем см. В разд, 3,3,1), если заданный адрес совпадает с одним из перечисленных в списке маршрутов, то функция возвращает экземпляр класса resolvermatch, хранящий сведения об адресе и совпавшем с ним маршруте. Если программное разрешение не увенчалось успехом, ТО возбуждается исключение Resolver404, производное ОТ Http404, из того же модуля django.urls.

Класс resolvermatch поддерживает следующие атрибуты:

□ func —ссылка на контроллер (функцию или класс);

□ kwargs — словарь со значениями URL-параметров, извлеченных из указанного адреса. Ключи элементов совпадают с именами URL-параметров. Если маршрут ^параметризованный — пустой словарь;

□ uri name — имя маршрута или None, если маршрут неименованный;

П route (начиная с Django 2.2) — строка с шаблонным путем;

□ view паше —то же самое, что и route, но только с добавлением имени или псевдонима приложения;

□ app name — строка с именем приложения или None, если таковое не задано;

П namespace — строка с псевдонимом приложения. Если псевдоним не указан, то атрибут хранит имя приложения.

Пример:

****

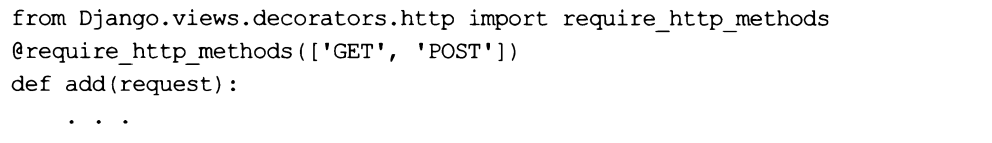


1. Дополнительные настройки контроллеров

Django предоставляет ряд декораторов, позволяющий задать дополнительные настройки контроллеров-функций.

Декораторы, задающие набор применимых к контроллеру HTTP-методов и объявленные В модуле django.views.decorators.http:

О require\_http\_methods ^последовательность обозначений методов>) —разрешает ДЛЯ контроллера только те HTTP-методы, обозначения которых указаны в заданной последовательности'.

****

□ require get () — разрешает для контроллера только метод GET;

□ require post () — разрешает для контроллера только метод POST;

□ require safe о — разрешает для контроллера только методы GET и HEAD (они считаются безопасными, т. К. Не изменяют внутренние данные сайта).

Если к контроллеру, помеченному одним из этих декораторов, отправить запрос с применением недопустимого HTTP-метода, то декоратор вернет экземпляр класса Httpresponsenotailowed, тем самым отправляя клиенту сообщение о недопустимом методе.

Декоратор gzip\_page() из модуля django.views.decorators.gzip сжимает ответ, сгенерированный помеченным контроллером, с применением алгоритма GZIP (конечно, если веб-обозреватель поддерживает такое сжатие).

1. Введение в контроллеры-классы

Контроллер-класс, в отличие от контроллера-функции, может самостоятельно выполнять некоторые утилитарные действия (выборку из модели записи по полученному ключу, вывод страницы и др.).

Основную часть функциональности контроллеры-классы получают из примесей (классов, предназначенных лишь для расширения функциональности других классов), от которых наследуют. Мы рассмотрим как примеси, так и полноценные классы.

Контроллер-класс записывается в маршруте не в виде ссылки, как контроллер-функция, а в виде результата, возвращенного методом as view (), который поддерживается всеми контроллерами-классами. Вот пример указания в маршруте контроллера-класса createview:

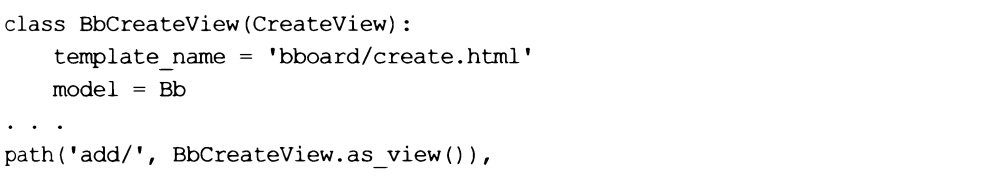


В вызове метода as viewo можно задать параметры контроллера-класса. Пример.

Указания модели и пути к шаблону (параметры model и template name соответственно):



Задать параметры контроллера-класса можно и по-другому: создав производный от него класс и указав параметры в его атрибутах:

****

Второй подход позволяет более радикально изменить поведение контроллера-класса, переопределив его методы, поэтому применяется чаще.

1. Базовые контроллеры-классы

Самые простые и низкоуровневые контроллеры-классы, называемые базовыми.

Объявлены В модуле django.views.generic.base.

**Контроллер V/eiv:**

Диспетчеризация по HTTP-методу контроллер-класс view определяет HTTP-метод, посредством которого был выполнен запрос, и исполняет код, соответствующий этому методу.

Класс поддерживает атрибут http method names, хранящий список имен допустимых HTTP-методов. По умолчанию он хранит список [’get’, ’post’, ’put', 'Patch', 'delete', 'head', 'options', 'trace'], включающий все методы, поддерживаемые протоколом HTTP.

Класс также содержит четыре метода (помимо уже знакомого нам as view ()), которые переопределяются в подклассах:

□ setup (self, request, \*args, \*\*kwargs) (начиная c Django 2.2)— выполняется самым первым и инициализирует объект контроллера.

Здесь и далее в параметре request передается объект запроса (в виде экземпляра класса httprequest), в параметре kwargs — словарь со значениями именованных URL-параметров.

Параметр args остался в ’’наследство” от первых версий Django и служит для передачи списка со значениями неименованных URL-параметров. В Django 2.0 и более поздних версиях неименованные URL-параметры не поддерживаются, поэтому данный параметр не используется.

В изначальной реализации создает в контроллере следующие атрибуты:

• request — Запрос, представленный экземпляром класса Request;

• kwargs — словарь со значениями URL-параметров.

Переопределив этот метод, можно сохранить в объекте контроллера какие-либо дополнительные данные;

□ dispatch (self, request, \*args, \*\*kwargs)—обрабатывает полученный в параметре request запрос и возвращает ответ, представленный экземпляром класса Httpresponse или его подкласса. Выполняется после метода setup ().

В изначальной реализации извлекает обозначение HTTP-метода, вызывает одноименный ему метод класса: get () — если запрос был выполнен НТТР-методом GET, post () — если запрос выполнялся методом POST, и т. П. — передавая ему все полученные параметры. Вызываемые методы должны быть объявлены в формате:

<имя метода класса> (self, request, \*args, \*\*kwargs)

Если метод класса, одноименный с HTTP-методом, отсутствует, ничего не делает. Единственное исключение — HTTP-метод HEAD: при отсутствии метода Head () вызывается метод get ();

□ http\_method\_not\_allowed (self, request, \*args, \*\*kwargs) — вызывается, если запрос был выполнен с применением неподдерживаемого НТТР-метода.

В изначальной реализации возвращает ответ типа httpresponsenotaiiowed со списком допустимых методов;

□ options (self, request, \*args, \*\*kwargs) —обрабатывает Запрос, выполненный HTTP-методом OPTIONS.

В изначальной реализации возвращает ответ с заголовком Allow, в котором записаны все поддерживаемые НТТР-методы.

Класс view используется крайне редко— обычно применяются производные от него контроллеры-классы, выполняющие более сложные действия.

**Примесь contextmixim**

Создание контекста шаблона класс-примесь contextmixin добавляет контроллеру-классу средства для формирования контекста шаблона:

□ extra context — атрибут, задающий содержимое контекста шаблона. Его значение должно представлять собой словарь, элементы которого будут добавлены в контекст;

□ get context data (self, \*\*kwargs) — метод, должен создавать и возвращать контекст шаблона. С параметром kwargs передается словарь, элементы которого должны быть добавлены в контекст шаблона.

В изначальной реализации создает пустой контекст шаблона, добавляет в него элемент view, хранящий ссылку на текущий экземпляр контроллера-класса, элементы ИЗ словарей kwargs И extra context.

**Примесь templateresponsemixin:**

Рендеринг шаблона класс-примесь templateresponsemixin добавляет наследующему классу средства для рендеринга шаблона.

На заметку!

Эта примесь и все наследующие от нее классы формируют ответ в виде экземпляра класса templateresponse, описанного В.

Поддерживаются следующие атрибуты и методы:

□ template name — атрибут, задающий путь к шаблону в виде строки;

□ get tempiate names (self) — метод, должен возвращать список путей к шаблонам, заданных в виде строк.

В изначальной реализации возвращает список из одного элемента— пути к шаблону, извлеченного ИЗ атрибута template name;

□ response class — атрибут, задает ссылку на класс, представляющий ответ (по умолчанию: templateresponse);

□ content type — атрибут, задающий MIME-тип ответа и его кодировку. По умолчанию — None (используется MIME-тип и кодировка по умолчанию);

□ render\_to\_response(self, context, \*\*response\_kwargs)—возвращает экземпляр класса, представляющего ответ. В параметре context передается контекст шаблона в виде словаря, а в параметре response kwargs — словарь, элементы которого будут переданы конструктору класса ответа в качестве дополнительных параметров.

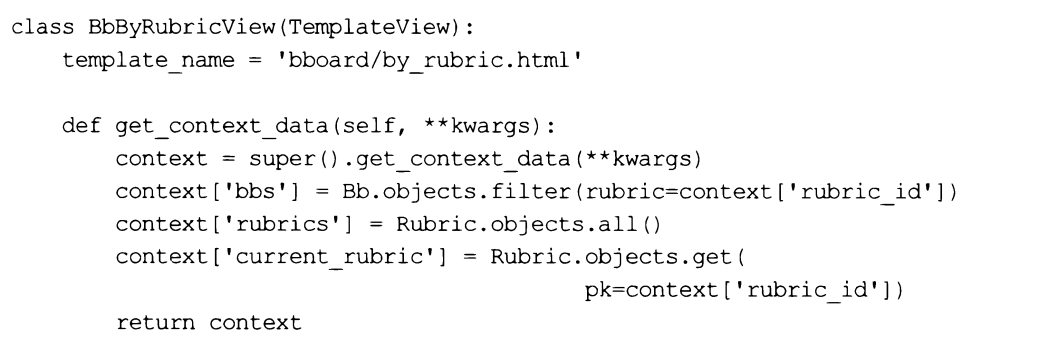
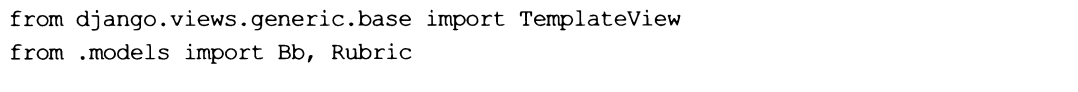
В изначальной реализации создает и возвращает экземпляр класса, указанного в атрибуте response\_class.

**Контроллер templateview**

Все вместе контроллер-класс Templateview наследует классы View, contextmixin и Templateresponsemixin. Он автоматически выполняет рендеринг шаблона и отправку ответа при получении запроса по методу GET.

В формируемый контекст шаблона добавляются все URL-параметры, которые присутствуют в маршруте, под своими изначальными именами.

Класс Templateview уже можно применять в практической работе. Приведен КОД ПРОИЗВОДНОГО ОТ Него контроллера-класса bbbyrubricview, который выводит страницу с объявлениями из выбранной рубрики.



Поскольку класс Templateview добавляет в контекст шаблона значения всех полученных им URL-параметров, мы можем извлечь ключ рубрики, обратившись к элементу rubric id контекста в переопределенном методе get context data ().

В классах, рассматриваемых далее, такой ’’номер” уже не пройдет, поскольку они не наследуют ОТ Templateview.