**Тема занятия № 23: Сигналы. Собственная модель пользователя**

1. Обработка сигналов

Сигнал сообщает о выполнении Django какого-либо действия: создании новой записи в модели, удалении записи, входе пользователя на сайт, выходе с него и пр.

К сигналу можно привязать обработчик — функцию или метод, который будет вызываться при возникновении сигнала.

Сигналы предоставляют возможность вклиниться в процесс работы самого фреймворка или отдельных приложений — неважно, стандартных или написанных самим разработчиком сайта— и произвести какие-либо дополнительные действия. Скажем, приложение django-cieanup, рассмотренное нами ранее и удаляющее ненужные файлы, чтобы отследить момент правки или удаления записи, обрабатывает Сигналы post\_init, pre\_save, post\_save И post\_delete.

**Обработка сигналов**

Все сигналы в Django представляются экземплярами класса signal или его подклассов. Этот класс поддерживает два метода, предназначенные для привязки к сигналу обработчика или отмены его привязки.

Для привязки обработчика к сигналу применяется метод connect () класса signal:

Connect(<обработчик>[, sender=None][, weak=True][, dispatch\_uid=None]) обработчик сигнала, как было сказано ранее, должен представлять собой функцию или метод. Формат написания этой функции (метода) мы рассмотрим позднее.

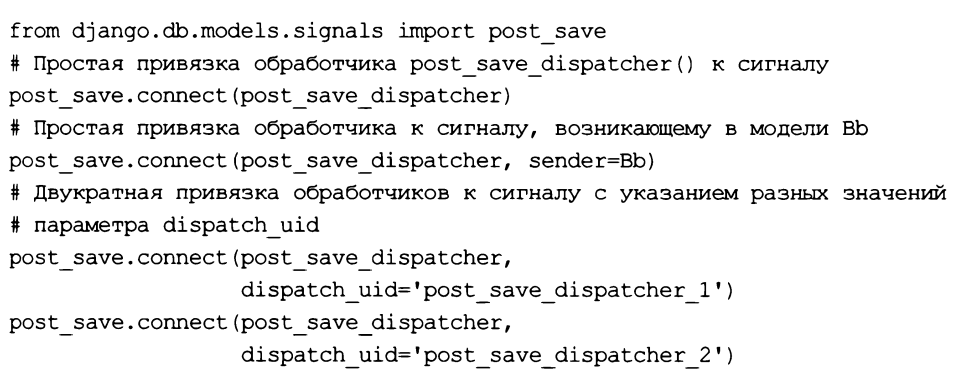
В необязательном параметре sender можно указать класс объекта, из которого отправляется текущий сигнал {отправителя). После чего обработчик будет обрабатывать сигналы исключительно от отправителей, относящихся к этому классу.

Если необязательному параметру weak присвоено значение True (а это его значение по умолчанию), то для связи отправителя и обработчика будет использована слабая ссылка Python, если присвоено значение False — обычная. Давать этому параметру значение False следует в случае, если после удаления всех отправителей нужда в обработчике пропадает — тогда он будет автоматически выгружен из памяти.

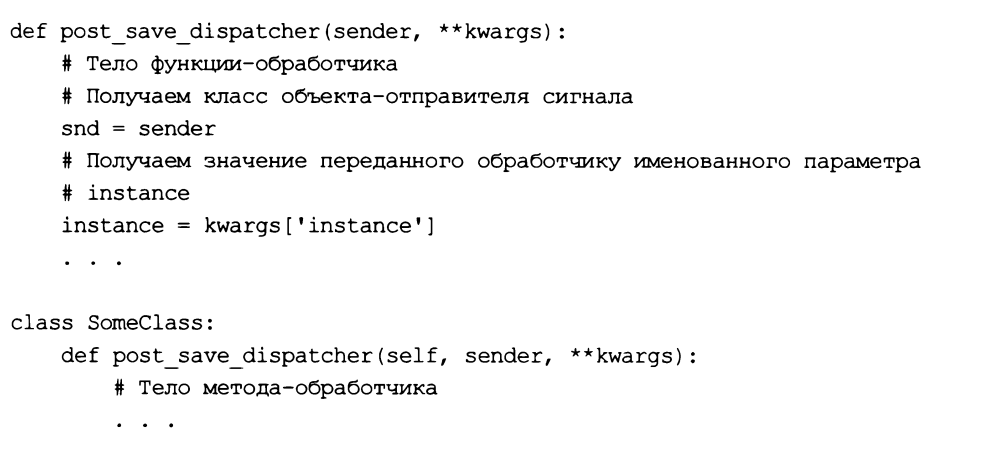
Необязательный параметр dispatch\_uid указывается, если к одному и тому же сигналу несколько раз привязывается один и тот же обработчик, и возникает необходимость как-то отличить одну такую привязку от другой. В этом случае в разных вызовах метода connect () нужно указать разные значения этого параметра, которые должны представлять собой строки.

Если к одному и тому же сигналу привязано несколько обработчиков, то они будут выполняться один за другим в той последовательности, в которой были привязаны к сигналу.

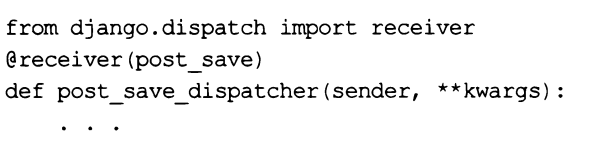
Рассмотрим несколько примеров привязки обработчика к сигналу post save, возникающему после сохранения записи модели:

****

Обработчик — функция или метод — должен принимать один позиционный параметр, с которым передается класс объекта-отправителя сигнала. Помимо этого, обработчик может принимать произвольное число именованных параметров, набор которых у каждого сигнала различается (стандартные сигналы Django и передаваемые ими параметры мы рассмотрим позже). Вот своего рода шаблоны для написания обработчиков разных типов:

****

Вместо метода connect о объекта сигнала можно использовать декоратор Receiver (<сигнал>), объявленный В модуле django.dispatch:

****

Код, выполняющий привязку к сигналам обработчиков, которые должны действовать все время, пока работает сайт, обычно записывается в модуле apps.py или Models.py.

Отменить привязку обработчика к сигналу позволяет метод disconnect () класса Signal:

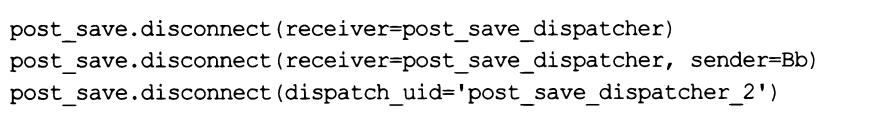


В параметре receiver указывается обработчик, ранее привязанный к сигналу. Если

Этот обработчик был привязан к сигналам, отправляемым конкретным классом, то

Последний следует указать в параметре sender. Если в вызове метода connect (), выполнившем привязку обработчика, был задан параметр dispatch uid с каким-либо

Значением, то удалить привязку можно, записав в вызове метода disconnect () только параметр dispatch uid и указав в нем то же значение. Примеры:



1. Встроенные сигналы Django

Сигналы, отправляемые подсистемой доступа к базам данных и объявленные в модуле Django.db.models.signals:

□ pre init — отправляется в самом начале создания новой записи модели, перед выполнением конструктора ее класса. Обработчику передаются следующие параметры:

• sender —класс модели, запись которой создается;

• args —список позиционных аргументов, переданных конструктору модели;

• kwargs — словарь именованных аргументов, переданных конструктору модели.

Например, при создании нового объявления выполнением выражения:



Обработчик с параметром sender получит ссылку на класс модели вь, с параметром args — ’’пустой” СПИСОК, а С параметром kwargs — словарь { ’title’ : ’Дом’, ’Content’: ’Трехэтажный, кирпич’, ’price’: 50000000};

□ post init — отправляется в конце создания новой записи модели, после выполнения конетруктора ее класса. Обработчику передаются следующие параметры:

• sender —класс модели, запись которой была создана;

• instance —объект созданной записи;

□ pre save — отправляется перед сохранением записи модели, до вызова ее метода save (). Обработчику передаются параметры:

• sender —класс модели, запись которой сохраняется;

• instance —объект сохраняемой записи;

• raw — True, если запись будет сохранена как есть, без обращения к другим записям за дополнительными данными и без исправления других записей, и False — в противном случае;

• update fieids — множество имен полей, заданных в параметре update\_fieids метода save (), или None, если этот параметр не был указан;

□ post save— отправляется после сохранения записи модели, после вызова ее метода save (). Обработчику передаются такие параметры:

• sender —класс модели, запись которой была сохранена;

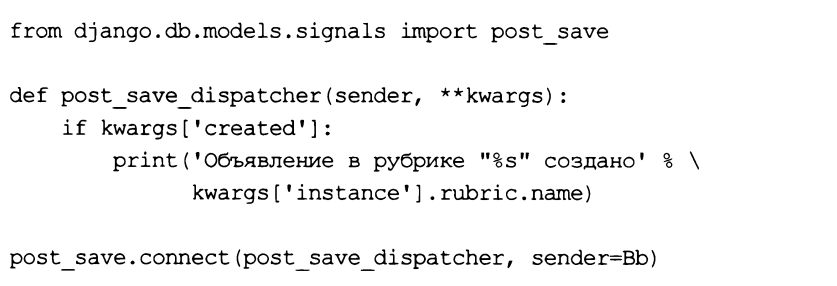
• instance —объект сохраненной записи;

• created— True, если это вновь созданная запись, и False — в противном случае;

• raw— True, если запись была сохранена как есть, без обращения к другим записям за дополнительными данными и без исправления других записей, и False — в противном случае;

• update fieids — множество имен полей, заданных в параметре update fieids метода save (), или None, если этот параметр не был указан.

Вероятно, это один из наиболее часто обрабатываемых сигналов. Вот пример его обработки с целью вывести в консоли Django сообщение о добавлении объявления:



□ pre delete — отправляется перед удалением записи, до вызова ее метода Delete о. Параметры, передаваемые обработчику:

• sender —класс модели, запись которой удаляется;

• instance —объект удаляемой записи;

□ post delete — отправляется после удаления записи, после вызова ее метода Delete (). Обработчик получит следующие параметры:

• sender — класс модели, запись которой была удалена;

• instance — объект удаленной записи. Отметим, что эта запись более не существует в базе данных;

□ m2m\_changed— отправляется связующей моделью при изменении состава записей моделей, связанных посредством связи ’’многие-со-многими”.

Связующая модель может быть явно задана в параметре through конструктора класса manytomanyfieid или же создана фреймворком неявно. В любом случае связующую модель можно получить из атрибута through объекта поля типа Manytomanyfieid. Пример привязки обработчика к сигналу, отправляемому связующей моделью, которая была неявно создана при установлении связи ”многие-со-многими” между моделями Machine и spare:



Обработчик этого сигнала принимает параметры:

• sender — класс связующей модели;

• instance — объект записи, в котором выполняются манипуляции по изменению состава связанных записей (т. Е. У которого вызываются методы add о, Create (), set () и др;

• action — строковое обозначение выполняемого действия:

D "pre add" — начало добавления новой записи в состав связанных;

° "post add" — окончание добавления новой связанной записи в состав связываемых;

D "pre remove" — начало удаления записи из состава связанных;

D "post remove" — окончание удаления записи из состава связанных;

D "pre ciear" — начало удаления всех записей из состава связанных;

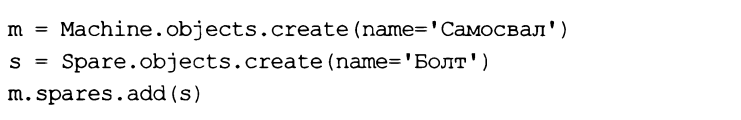
° "post ciear” — окончание удаления всех записей из состава связанных;

• reverse— False, если изменение состава связанных записей выполняется в записи ведущей модели, и True, если в записи ведомой модели;

• model — класс модели, к которой принадлежит запись, добавляемая в состав связанных или удаляемая оттуда;

• pk set — множество ключей записей, добавляемых в состав связанных или удаляемых оттуда. Для действий "pre clear" и "post clear" всегда None.

Например, при выполнении операций:



Обработчик сигнала m2m\_changed с параметром sender получит ссылку на класс промежуточной модели (у нас — созданной самим фреймворком), с параметром Instance — запись m (т. к. действия по изменению состава связанных записей выполняются в ней), с параметром action— строку "pre\_add”, с параметром Reverse — False (действия по изменению состава связанных записей выполняются в записи ведущей модели), с параметром model — модель spare, а с параметром pk set— множество из единственного элемента— ключа записи s.

Впоследствии тот же самый сигнал будет отправлен еще раз, и его обработчик получит с параметрами те же данные, за исключением параметра action, который будет иметь значение "post add”. А после выполнения действия:



Обработчик сигнала m2m\_changed с параметром sender получит ссылку на класс промежуточной модели, с параметром instance — запись s, с параметром Action — строку "pre remove", С параметром reverse — True (поскольку теперь действия по изменению состава связанных записей выполняются в записи ведомой модели), С параметром model — модель Machine, а С параметром pk set — множество из единственного элемента — ключа записи ш. Далее тот же самый сигнал будет отправлен еще раз, и его обработчик получит с параметрами те же данные, за исключением параметра action, который будет иметь значение ”post\_remove".

Сигналы, отправляемые подсистемой обработки запросов и объявленные в модуле Dj ango.core.signals:

□ request started — отправляется в самом начале обработки запроса. Обработчик получит параметры:

• sender— класс django.core.handlers.wsgi.wsgihandler, обрабатывающий все полученные запросы;

• environ — словарь, содержащий переменные окружения;

□ request finished— отправляется после пересылки ответа клиенту. Обработчик с параметром sender получит класс dj ango. Core. Handlers . Wsgi . Wsgihandler, обрабатывающий все полученные запросы;

□ got request exception — отправляется при возбуждении исключения в процессе обработки запроса. Вот параметры, передаваемые обработчику:

• sender — None;

• request — Сам Запрос В виде экземпляра класса httprequest. сигналы, отправляемые подсистемой разграничения доступа и объявленные в модуле django.contrib.auth.signals:

□ user iogged in — отправляется после удачного входа на сайт. Параметры, передаваемые обработчику:

• sender — класс модели пользователя (user, если не была задана другая модель);

• request — текущий запрос, представленный экземпляром класса httprequest;

• user — запись пользователя, который вошел на сайт;

□ user iogged out — отправляется после удачного выхода с сайта. Вот параметры, которые получит обработчик:

• sender— класс модели пользователя или None, если пользователь ранее не выполнил вход на сайт;

• request — текущий запрос в виде экземпляра класса httprequest;

• user — запись пользователя, который вышел с сайта, или None, если пользователь ранее не выполнил вход на сайт;

□ user iogin faiied— отправляется, если посетитель не смог войти на сайт. Параметры, передаваемые обработчику:

• sender — строка с именем модуля, выполнявшего аутентификацию;

• credentials— словарь со сведениями, занесенными посетителем в форму входа и переданными впоследствии функции authenticate (). Вместо пароля будет подставлена последовательность звездочек;

• request — текущий запрос в виде экземпляра класса httprequest, если таковой был передан функции authenticate (), В противном случае — None.

На заметку!

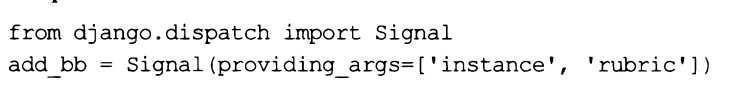
Некоторые специфические сигналы, используемые внутренними механизмами Django или подсистемами можно найти на странице https://docs.djangoproject.eom/en/3.0/ref/signals/.

1. Объявление своих сигналов

Сначала нужно объявить сигнал, создав экземпляр класса signal из модуля Django.dispatch. Конструктор этого класса вызывается согласно формату:

Signal(providing\_args=<cnncok имен параметров, передаваемых обработчику^имена параметров в передаваемом списке должны быть представлены в виде строк.

Пример объявления сигнала add bb, который будет передавать обработчику параметры instance И rubric:



Для отправки объявленного сигнала применяются два следующих метода класса Signal:

О send. (<отправитель> [, <именованные параметры, указанные при объявлении сигнала^ )— выполняет отправку текущего сигнала от имени указанного отправителя, возможно, с именованными параметрами, которые были указаны при объявлении сигнала и будут отправлены его обработчику.

В качестве результата метод возвращает список, каждый из элементов которого представляет один из привязанных к текущему сигналу обработчиков. Каждый элемент этого списка представляет собой кортеж из двух элементов: ссылки на обработчик и возвращенный им результат. Если обработчик не возвращает результата, то вторым элементом станет значение None. Пример:

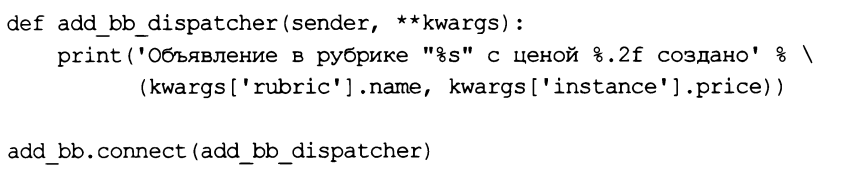


Если к сигналу привязано несколько обработчиков и в одном из них было возбуждено исключение, последующие обработчики выполнены не будут;

□ send\_robust (<отправитель>[, <именованные параметры, указанные при объявлении сигнала>]) — то же самое, что send (), но обрабатывает все исключения, что могут быть возбуждены в обработчиках. Объекты исключений будут присутствовать в результате, возвращенном методом, во-вторых, элементах соответствующих вложенных кортежей.

Поскольку исключения обрабатываются внутри метода, то, если к сигналу привязано несколько обработчиков и в одном из них было возбуждено исключение, последующие обработчики все же будут выполнены.

Объявленный нами сигнал может быть обработан точно так же, как и любой из встроенных в Django:

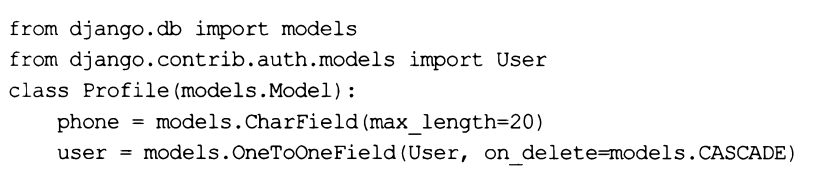
****

1. Создание своей модели пользователя

Для хранения списка пользователей в подсистеме разграничения доступа Django предусмотрена стандартная модель user, объявленная в модуле django.contrib.

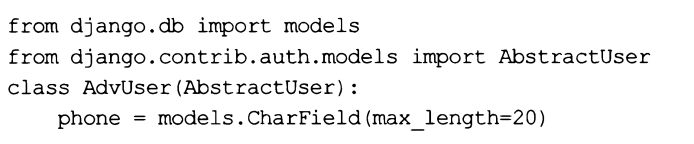
**Auth.models.** Эта модель хранит объем сведений о пользователе, вполне достаточный для многих случаев. Однако часто приходится сохранять в составе сведений о пользователе дополнительные данные: номер телефона, интернет-адрес сайта, признак, хочет ли пользователь получать по электронной почте уведомления о новых сообщениях, и т. п.

Можно объявить дополнительную модель, поместить в нее поля для хранения всех нужных данных и добавить поле, устанавливающее связь ”один-с-одним” со стандартной моделью пользователя. Вот пример создания подобной дополнительной модели:



Разумеется, при создании нового пользователя придется явно создавать связанную с ним запись модели, хранящую дополнительные сведения. Зато не будет никаких проблем с подсистемой разграничения доступа и старыми дополнительными библиотеками, поскольку для хранения основных сведений о пользователях будет использоваться стандартная модель user.

Другой подход заключается в написании своей собственной модели пользователя. Такую модель следует сделать производной от класса Abstractuser, который объявлен в модуле django.contrib.auth.models, реализует всю функциональность по хранению пользователей и представляет собой абстрактную модель — собственно, класс стандартной модели пользователей user также является производным от класса Abstractuser. Пример:

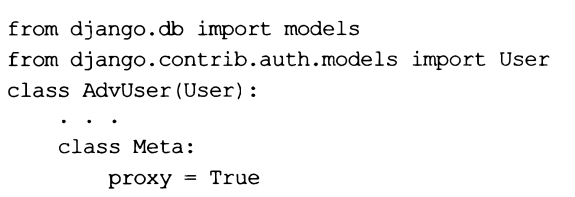
****

Новую модель пользователя следует указать в параметре auth user model настроек проекта.



В таком случае не придется самостоятельно создавать связанные записи, хранящие дополнительные сведения о пользователе — это сделает Django. Однако нужно быть готовым к тому, что некоторые дополнительные библиотеки, в особенности старые, не считывают имя модели пользователя из параметра auth user model, а обращаются напрямую к модели user. Если такая библиотека добавит в список нового пользователя, то он будет сохранен без дополнительных сведений, и код, использующий эти сведения, не будет работать.

Если нужно лишь расширить или изменить функциональность модели пользователя, то можно создать на его основе прокси-модель, также не забыв занести ее в параметр

****

И наконец, можно написать полностью свой класс модели. Однако такой подход применяется весьма редко из-за его трудоемкости. Интересующиеся могут обратиться к странице <https://docs.djangoproject.com/eii/3.0/topics/aiith/custoimzing/>, где приводятся все нужные инструкции.