**Тема занятия № 55: Модуль 32. Журналирование и отладка**

1. Средства журналирования

Django предоставляет ряд отладочных инструментов, которые пригодятся как при отладке разрабатываемого сайта, так и в процессе его эксплуатации, для фиксации возникающих ошибок.

**Средства журналирования**

Журналирование — это фиксация каких-либо событий, происходящих в работающем сайте, обычно — ошибок, возникающих в его коде. Подсистема журналирования, встроенная в Django, может выводить сообщения о произошедших событиях на экран, отправлять их по электронной почте или записывать в файл журнала.

**Настройка подсистемы журналирования**

Настройки подсистемы журналирования записываются в модуле settings.py пакета конфигурации. Все они указываются в параметре logging в виде словаря, ключи элементов которого задают названия различных параметров, а значения элементов — значения этих параметров.

Вот параметры, доступные для указания:

□ version— номер версии стандарта, в котором записываются настройки подсистемы журналирования, в виде целого числа. На данный момент поддерживается только версия 1;

□ formatters — перечень доступных для использования форматировщиков. Форматировщик определяет формат сообщений о произошедших событиях, выводимых подсистемой;

□ filters — перечень доступных фильтров сообщений. Фильтры отбирают для вывода только сообщения, удовлетворяющие определенным условиям;

□ handlers — перечень доступных обработчиков. Обработчики непосредственно выполняют вывод определенным способом (на консоль, в файл, по электронной почте и др.) Сообщений, прошедших через фильтры, в формате, заданном форматировщиком;

□ loggers — перечень доступных регистраторов. Регистратор собирает все сообщения, отправленные заданными подсистемами Django, и передает их указанным обработчикам для вывода;

□ disable existing loggers — если True, ТО регистраторы, используемые ПО ум0лчанию, работать не будут, если False —будут (по умолчанию — True).

**Объект сообщения**

Каждое сообщение о произошедшем событии представляется в виде экземпляра класса logrecord из модуля logging стандартной библиотеки Python.

Вот атрибуты класса logrecord, хранящие полезную для нас информацию о сообщении:

□ message —текст сообщения в виде строки;

□ levelname— обозначение уровня Сообщения В виде строки: ’’DEBUG", "INFO", "warning", "error" или "critical". Все эти уровни сообщений описаны в табл. 23.1;

□ leveino —обозначение уровня сообщения в виде целого числа;

□ pathname —полный путь выполняемого в данный момент файла в виде строки;

□ filename —имя выполняемого в данный момент файла в виде строки;

□ module— имя выполняемого в данный момент модуля, полученное из имени файла путем удаления у него расширения, в виде строки;

□ lineno— порядковый номер выполняемой в данный момент строки программного кода в виде целого числа;

□ funcname —имя выполняемой в данный момент функции в виде строки;

□ asctime —временная отметка создания сообщения в виде строки;

□ created— временная отметка создания сообщения в виде вещественного числа, представляющего собой количество секунд, что прошли с полуночи 1 января 1970 года. Для формирования этой величины применяется функция time о из модуля time Python;

□ msecs —миллисекунды из времени создания сообщения в виде целого числа;

□ reiativecreated — количество миллисекунд, прошедших между запуском регистратора и созданием текущего сообщения, в виде целого числа;

□ exc infо — кортеж из трех значений: ссылки на класс исключения, самого объекта исключения и объекта, хранящего стек вызова. Для формирования этого кортежа применяется функция exc infо () из модуля sys Python;

□ stack infо — объект, хранящий стек вызовов;

□ process — идентификатор процесса в виде целого числа (если таковой удается определить);

□ processname —имя процесса в виде строки (если таковое удается определить);

□ thread — идентификатор потока в виде целого числа (если таковой удается определить);

□ threadname —имя потока в виде строки (если таковое удается определить);

□ name —имя регистратора, оставившего это сообщение, в виде строки.

**Форматировщики**

Форматировщик задает формат, в котором представляется сообщение о произошедшем событии.

Перечень форматировщиков в параметре formatters указывается в виде словаря.

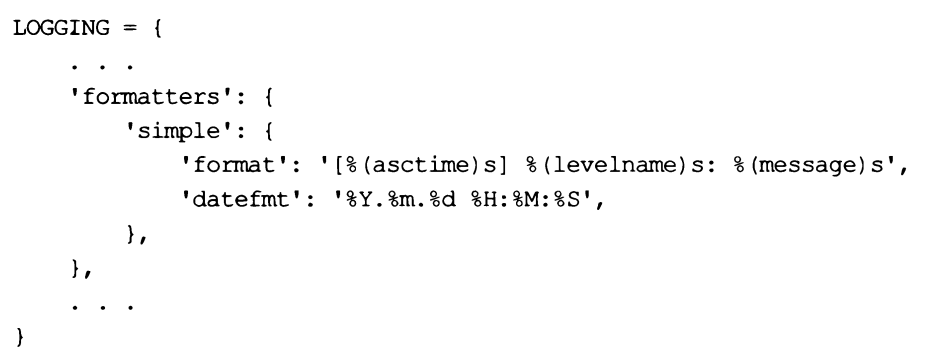
Ключами его элементов служат имена объявляемых форматировщиков, а значения элементов задают значения параметров соответствующих форматировщиков.

Доступны следующие параметры форматировщиков:

□ format— строка формата для формирования текста сообщения. Для вставки в текст значений атрибутов объекта сообщения применяются языковые КОНСТРУКЦИИ вида % (<имя атрибута^ s;

□ datefmt — строка формата для формирования временных отметок. В ней должны присутствовать специальные символы, поддерживаемые функцией strftimeo ИЗ модуля time. По умолчанию: ”%Y-%m-%d %Н:%М:%S,uuu".

Пример объявления простого форматировщика с именем simple:

****

Он ВЫВОДИТ сообщения В формате [<временная отметка>] <уровень>: <текст>, где временная отметка создания события имеет формат <год>. <месяц>. <число> <часы>:

<минуты>:<секунды>.

**Фильтры**

Фильтр отбирает для вывода только те сообщения, которые удовлетворяют определенным условиям.

Перечень фильтров записывается в таком же формате, как и перечень форматировщиков. У каждого объявленного фильтра следует задать обязательный параметр о (две круглые скобки), указывающий строку с именем класса фильтра. Если конструктор этого класса принимает какие-либо параметры, то они задаются там же — в настройках фильтра.

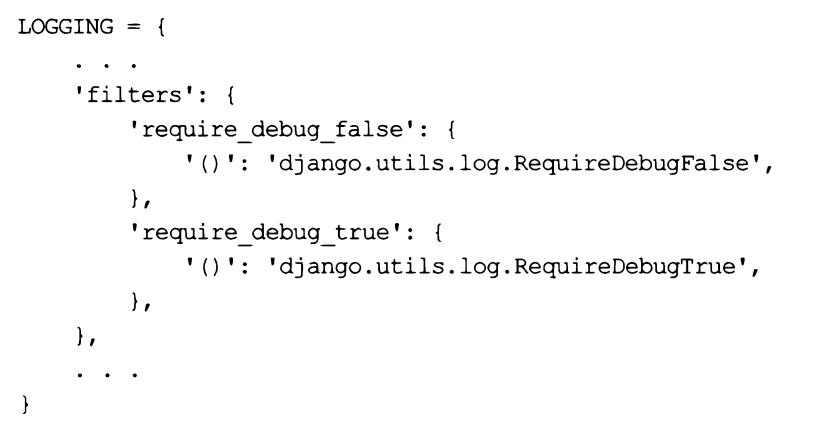
Все доступные классы фильтров объявлены в модуле django.utils.log:

□ requiredebugtrue — выводит сообщения только в том случае, если включен отладочный режим (параметру debug настроек проекта присвоено значение True).

Об отладочном и эксплуатационном режимах сайта рассказывалось ранее:

□ requiredebugfaise— выводит сообщения только в том случае, если включен эксплуатационный режим (параметру debug настроек проекта присвоено значение False).

Пример использования этих классов фильтров:

****

□ callbackfilter(callback=<функция>} — ВЫВОДИТ ТОЛЬКО те сообщения, ДЛЯ КОТОРЫХ указанная в параметре callback функция вернет True. Функция должна в качестве единственного параметра принимать сообщение, представленное экземпляром класса logrecord (см.разд 29.1.2).

Пример объявления фильтра info filter, отбирающего только сообщения уровня info:

****

**Обработчики**

Обработчики непосредственно выполняют вывод сообщений на поддерживаемые ими устройства: в командной строке, в файл или куда-либо еще.

Перечень обработчиков записывается в таком же формате, что и перечень форматировщиков. У каждого из обработчиков можно задать такие параметры:

□ class — строка с именем класса обработчика, который будет выполнять вывод сообщений. Поддерживаемые Django классы обработчиков будут рассмотрены позже;

□ level — минимальный уровень сообщений в виде строкового обозначения. Обработчик станет выводить сообщения, уровень которых не меньше заданного,

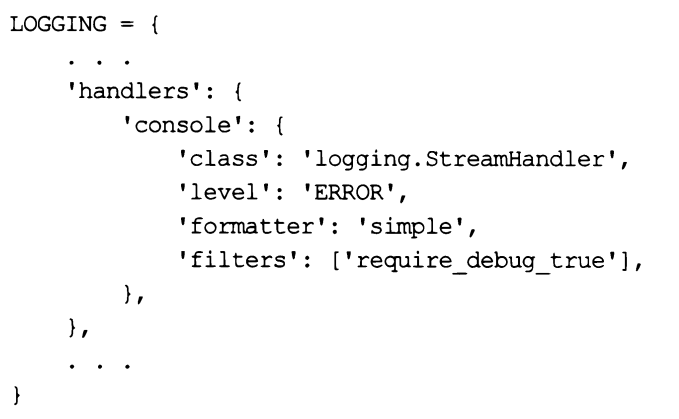
Сообщения меньшего уровня выводиться не будут. Если параметр не указан, то обработчик будет выводить сообщения всех уровней;

□ formatter— форматировщик, который будет применяться для формирования сообщений. Если параметр не указан, сообщения будут иметь формат по умолчанию: <текст сообщения^

□ filters — список фильтров, через которые будут проходить выводимые обработчиком сообщения. Чтобы сообщение было выведено, оно должно пройти через все включенные в список фильтры. Если параметр не указан, фильтры использоваться не будут;

□ именованные параметры, принимаемые конструктором класса обработчика (если таковые есть).

Пример указания фильтра, выводящего сообщения уровня error и выше на консоль с применением фильтра require\_debug\_true и форматировщика simple:

****

Вот наиболее часто используемые классы обработчиков:

□ logging. Streamhandler —ВЫВОДИТ сообщения В КОНСОЛИ;

О logging.filehandler(fиепате=<луть к файлу>[, mode='a'][, encoding=None][, deiay=Faise]) — сохраняет сообщения в файле с заданным путем. Размер получающегося файла не ограничен.

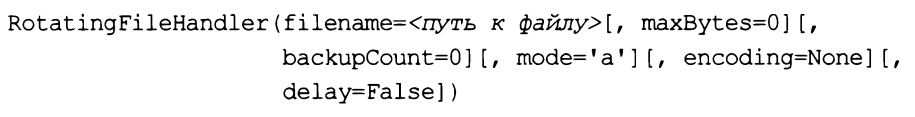
Параметр mode задает режим открытия файла; если он не указан, файл открывается для добавления. Параметр encoding указывает кодировку файла; если он опущен, то Python сам выберет кодировку.

Если параметру delay задать значение True, файл будет открыт только в момент вывода самого первого сообщения. Если же присвоить ему значение False, файл будет открыт непосредственно при инициализации класса обработчика (поведение по умолчанию).

Пример использования этого класса:

****

□ logging. Handlers. Rotatingfilehandler — TO же самое, ЧТО filehandler, НО вместо одного большого файла создает набор файлов ограниченного размера. Как только размер очередного файла приближается к указанному пределу, создается новый файл. Формат вызова конструктора этого класса:

****

Параметр maxbytes устанавливает размер файла, при превышении которого будет создан новый файл с сообщениями, в байтах. Если задать значение о, то класс обработчика будет сохранять все сообщения в один файл неограниченного размера, т. Е. Вести себя, как класс filehandler.

Параметр backupcount указывает количество ранее созданных файлов, которые будут сохраняться на диске. К расширениям этих файлов будут добавляться последовательно увеличивающиеся целые числа. Так, если сообщения записываются в файл django-site.log, предыдущие файлы получат имена django-site.log. 1, Django-site.log.2 и т. Д. Если количество таких файлов превысит заданную в параметре величину, наиболее старые файлы будут удалены.

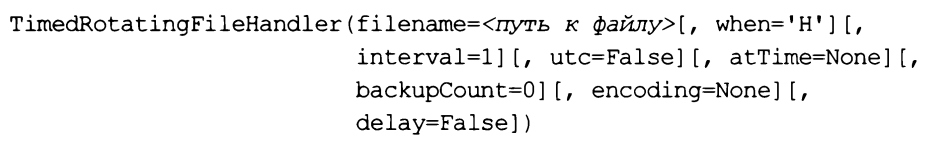
Если параметру backupcount присвоить значение о, то все сообщения станут сохраняться в один файл неограниченного размера (при этом значение параметра Maxbytes будет проигнорировано).

О назначении остальных параметров конструктора говорилось в описании класса filehandler.

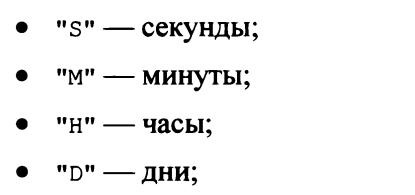
Пример:

****

□ logging.handlers.timedrotatingfilehandler — TO же самое, ЧТО rotatingfilehandler, только начинает запись в новый файл не при превышении указанного размера файла, а по прошествии заданного временного промежутка. Формат вызова конструктора:

****

Параметр when указывает разновидность промежутка времени, через который следует создать новый файл. Доступны значения:

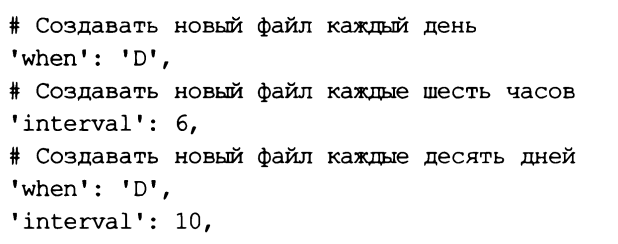
****

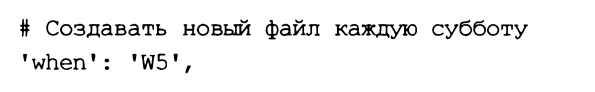
• "w<Howep дня недели>" — каждый день недели с указанным номером. В качестве номера дня недели нужно указать целое число от о (понедельник) до 6 (воскресенье);

• "midnight" — каждый день В ПОЛНОЧЬ.

Параметр interval задает количество промежутков времени заданной разновидности, после которых нужно начинать запись в новый файл (по умолчанию: 1).

Примеры:





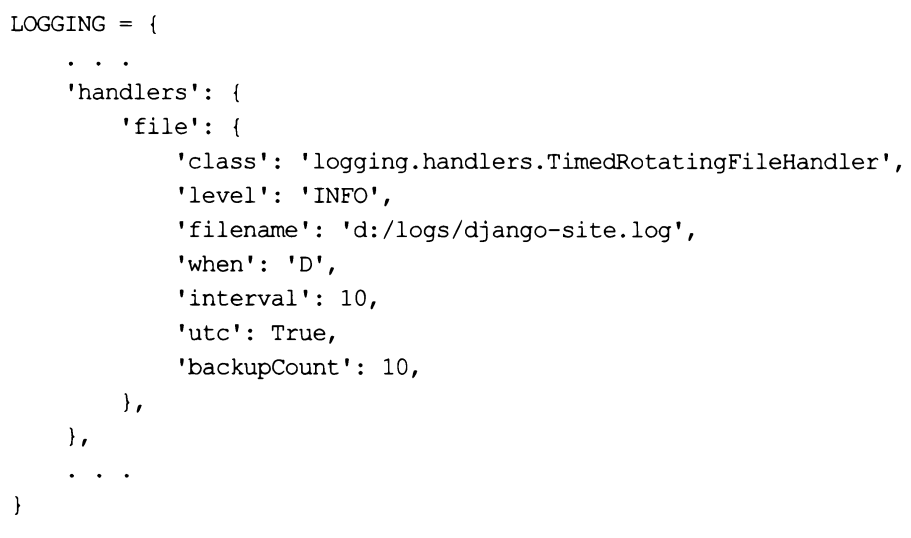
К расширениям ранее созданных файлов с сообщениями будут добавляться строки формата <год>-<месяц^<число>[\_<часы>-<минуты>-<секунды>], причем вторая половина, с часами, минутами и секундами, может отсутствовать, если задан временной интервал, превышающий один день.

Если параметру utc присвоить значение True, будет применяться всемирное координированное время (UTC). Присвоение значения False укажет Django использовать местное время.

Параметр attime принимается во внимание только в том случае, если параметру When дано значение "w<номер дня недели>" или ’’midnight". Значением параметра attime должна быть отметка времени в виде объекта типа time из модуля Datetime, которая укажет время, в которое следует начать запись в новый файл.

О назначении остальных параметров конструктора говорилось в описании классов filehandler И rotatingfilehandler.

Пример:

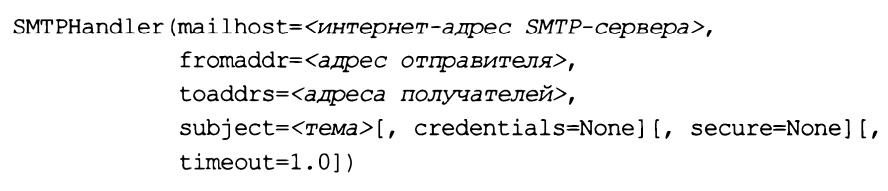


□ django.utils.log.adminemailhandler([include\_html=False][,][email\_backend=None]) — отправляет сообщения по электронной почте по адресам, приведенным в списке параметра admins настроек проекта.

Если параметру inciude htmi присвоить значение True, то в письмо будет вложена веб-страница с полным текстом сообщения об ошибке. Значение False приводит к отправке обычного сообщения.

Посредством параметра email backend можно выбрать другой класс, реализующий отправку электронных писем. Список доступных классов такого назначения, равно как и формат значения параметра, в описании параметра email backend настроек проекта.

□ logging. Handlers. Smtphandier —отправляет сообщения по электронной почте на произвольный адрес. Формат конструктора:

****

Интернет-адрес SMTP-сервера может быть задан В видов!

• строки — если сервер работает через стандартный ТСР-порт;

• кортежа из собственно интернет-адреса и номера ТСР-порта — если сервер работает через нестандартный порт.

Адреса получателей указываются В виде списка. Адрес отправителя И тему отправляемого письма нужно задать в виде строк.

Параметр credentials указывает кортеж из имени и пароля для подключения к SMTP-серверу. Если сервер не требует аутентификации, параметр нужно опустить.

Доступные значения для параметра secure:

• None — если протоколы SSL и TLS не используются (значение по умолчанию);

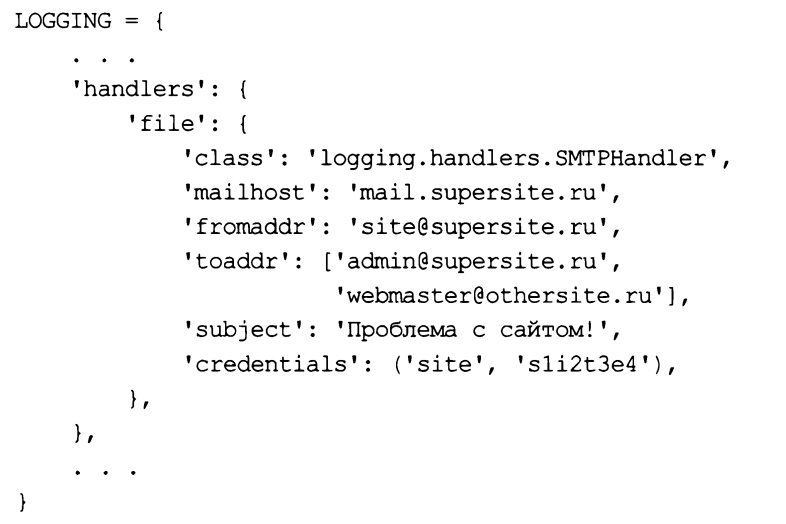
• ’’пустой” кортеж — если используется протокол SSL или TLS;

• кортеж из пути к файлу с закрытым ключом;

• кортеж из пути к файлу с закрытым ключом и пути к файлу с сертификатом.

Параметр timeout указывает промежуток времени (в виде целого числа в секундах), в течение которого класс-отправитель будет пытаться установить соединение с SMTP-сервером.

Пример:

****

□ logging.nuiihandier— вообще не выводит сообщения. Применяется для подавления вывода сообщений определенного уровня.

**Регистраторы**

Регистраторы занимаются сбором всех сообщений, отправляемых указанными подсистемами Django.

Перечень регистраторов записывается в виде словаря. В качестве ключей его элементов указываются имена регистраторов, а значениями элементов должны быть словари, задающие настройки этих регистраторов.

Django поддерживает следующие регистраторы:

□ django —собирает сообщения от всех подсистем фреймворка;

□ django.request— собирает сообщения от подсистемы обработки запросов и формирования ответов. Ответы с кодами статуса 5хх создают сообщения с уровнем error, сообщения с кодами 4хх —сообщения уровня WARNING.

Объект сообщения, в дополнение к приведенным получит следующие атрибуты:

• status code — числовой код статуса ответа;

• request —объект запроса;

□ django.server— то же самое, что django.request, но работает только под отладочным веб-сервером Django;

□ django.template— собирает сообщения об ошибках, присутствующих в коде шаблонов. Такие сообщения получают уровень debug;

□ django.db.backends— собирает сообщения обо всех операциях с базой данных сайта. Такие сообщения получают уровень debug.

Объект сообщения, в дополнение к приведенным получит следующие атрибуты:

• sqi —SQL-код команды, отправленной СУБД;

• duration —продолжительность выполнения этой команды;

• params —параметры, переданные вместе с этой командой;

□ django.db.backends.schema— собирает сообщения обо всех операциях, производимых над базой данных в процессе выполнения миграций.

Объект сообщения, в дополнение к приведенным получит следующие атрибуты:

• sqi —SQL-код команды, отправленной СУБД;

• params — параметры, переданные вместе с этой командой;

□ django.security. <класс исключения> — собирает сообщения о возникновении исключений указанного класса. Поддерживаются только класс исключения Suspiciousoperation И все его подклассы (disallowedhost, disallowedmodeladminlookup, disallowedmodeladmintofield, disallowedredirect, invalidsessionkey, Requestdatatoobig, suspiciousfileoperation, suspiciousmultipartform, suspicioussession И toomanyfieldssent);

□ django.security.csrf — собирает сообщения о несовпадении электронных жетонов безопасности, указанных в веб-формах посредством тега csrf token, с ожидаемыми.

Параметры, поддерживаемые всеми регистраторами:

□ handlers— список обработчиков, которым регистратор будет пересылать собранные им сообщения для вывода;

□ propagate — если True, то регистратор будет передавать собранные сообщения более универсальным регистраторам (обычно это регистратор django). Если False, то сообщения передаваться не будут. По умолчанию — False;

□ level — минимальный уровень сообщений в виде строкового обозначения.

Регистратор станет собирать сообщения, уровень которых не меньше заданного, сообщения меньшего уровня будут отклоняться. Если параметр не указан, регистратор будет собирать сообщения всех уровней.

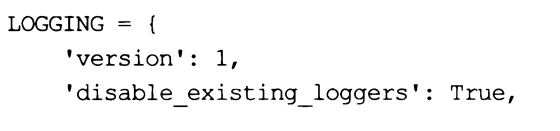
У универсального регистратора, принимающего сообщения от регистраторов более специализированных, значение параметра level, судя по всему, во внимание не принимается. Следовательно, он собирает сообщения любого уровня;

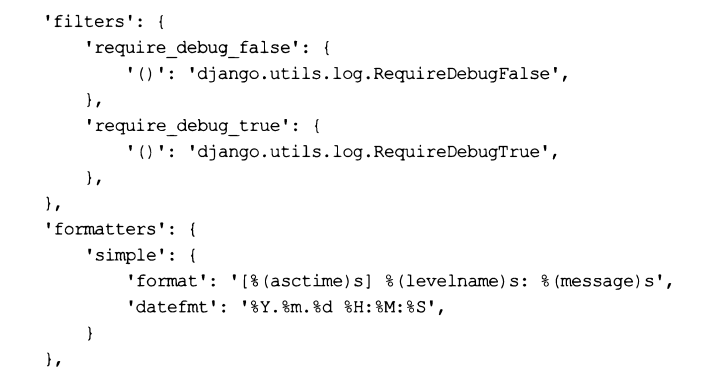
□ filters — список фильтров, через которые будут проходить собираемые регистратором сообщения. Чтобы сообщение было воспринято, оно должно пройти через все включенные в список фильтры. Если параметр не указан, фильтры использоваться не будут.

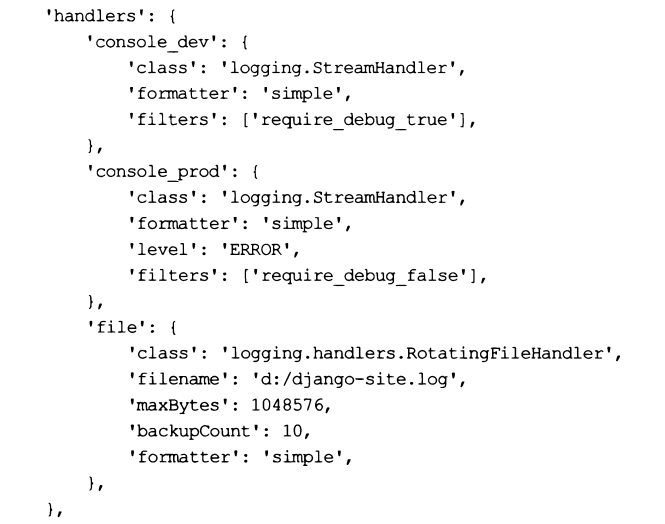
**Пример настройки**

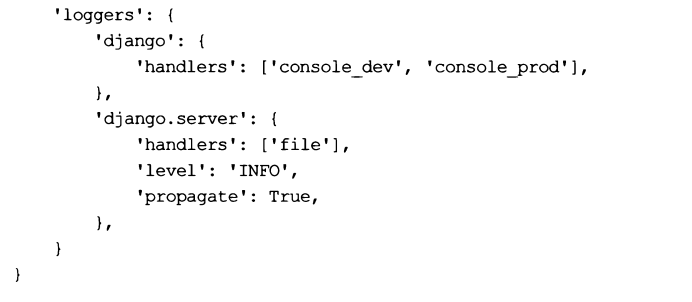
**Подсистемы журналирования**

Приведен пример кода, задающего настройки подсистемы журналирования, который можно использовать на практике. Эти настройки записываются модуле settings.py пакета конфигурации.









Все регистраторы, используемые по умолчанию, отключены указанием у параметра Disabie\_existing\_ioggers значения True. Фактически проект задействует свои собственные средства журналирования.

Объявлены два фильтра: require\_debug\_faise, пропускающий сообщения только в эксплуатационном режиме, и require\_debug\_true, который будет пропускать сообщения только в отладочном режиме.

Форматировщик simple ВЫВОДИТ сообщения В формате [<временная отметка создания>] <уровень>: <текст>.

Обработчиков в нашей конфигурации целых три:

□ consoie dev— выводит в командной строке сообщения любого уровня, прошедшие через фильтр require\_debug\_true, посредством форматировщика Simple;

□ consoie prod — выводит в командной строке сообщения уровня error, прошедшие через фильтр require\_debug\_false, посредством форматировщика simple;

□ file— сохраняет в файл d:\django-site.log сообщения любого уровня посредством форматировщика simple. При превышении файлом размера в 1 Мбайт (1 048 576 байт) будет создан новый файл. Всего будет одновременно храниться 10 таких файлов с сообщениями.

Наконец, объявлены два регистратора:

□ django— универсальный, собирает сообщения из всех подсистем фреймворка И ВЫВОДИТ ИХ посредством обработчиков consoie dev И consoie prod;

□ django.server — собирает сообщения уровней info и выше от подсистемы обработки запросов, когда запущен отладочный веб-сервер, и выводит их через обработчик file.

В результате, если включен отладочный режим, в командной строке будут выводиться все сообщения, а при активном эксплуатацйонном режиме — только сообщения о критических ошибках. А сообщения от подсистемы обработки запросов в любом случае будут дополнительно записываться в файл.

1. Средства отладки

**Веб-страница сообщения об ошибке**

Если при исполнении программного кода произошла ошибка, то Django выведет стандартную веб-страницу с соответствующим сообщением.

Содержимое страницы разделено на отдельные области, представляющие различную информацию:

□ Общие сведения об ошибке — наиболее важная, выделенная на странице желтым фоном. В ней присутствуют следующие полезные для нас сведения:

• имя класса исключения, возбужденного при возникновении ошибки, и запрошенный клиентом интернет-адрес;

• текстовое описание ошибки;

• Request Method — HTTP-метод, посредством которого был выполнен запрос;

• Request URL — полный интернет-адрес, запрошенный клиентом, с указанием домена и номера порта;

• Exception Туре — имя класса исключения;

• Exception Value — текстовое описание ошибки;

• Exception Location — полный путь к программному файлу, в коде которого была допущена ошибка, с указанием номера строки кода.

Также в этой области выводятся номера версий Django и Python, путь к файлу исполняющей среды Python, список путей, по которым исполняющая среда ищет библиотеки, и время обнаружения ошибки.

□ Traceback— стек вызовов. Организован в виде набора разделов, в каждом из которых выводятся полный путь к программному файлу, строка исходного кода и список локальных переменных с их значениями.

Каждая строка исходного кода в таком разделе представляет собой заголовок спойлера (раскрывающейся панели). Если щелкнуть на расположенном в его правой части многоточии, появится фрагмент исходного кода, в котором находится эта строка.

Во вложенных спойлерах с заголовками Local vars перечислены все локальные переменные и их значения.

□ Request information — сведения о полученном запросе:

• USER — имя текущего пользователя или anonymoususer, если это гость;

• GET — GET-параметры и их значения;

• POST — POST-параметры и их значения;

• FILES — отправленные посетителем файлы;

• COOKIES — cookie и их значения;

• МЕТА — значения заголовков запроса, сведения об исполняющей среде Python и операционной системе;

• Settings — настройки проекта.

Внимание!

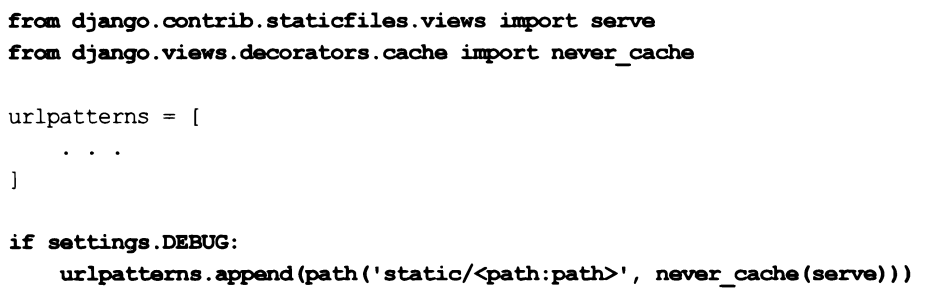
Описанная здесь страница со сведениями об ошибке выводится только в том случае, если активен отладочный режим. Если сайт переведен в эксплуатационный режим, будет выведена обычная страница с сообщением об ошибке 503 (внутренняя ошибка сервера).

**Отключение кэширования статических файлов**

Часто в процессе программирования сайта также дорабатываются используемые им таблицы стилей. Но есть проблема: при отправке клиенту статических сайтов Django устанавливает для них очень большое время кэширования на стороне клиента. И если исправить таблицу стилей и перезагрузить страницу, то веб обозреватель использует кэшированную старую копию таблицы стилей, и мы не увидим на странице никаких изменений.

В качестве решения можно очищать кэш веб-обозревателя после каждого изменения таблиц стилей или вообще отключить кэширование. Но удобнее указать Django, чтобы он запрещал клиенту кэшировать статические файлы. Для этого достаточно выполнить всего три простых действия:

□ добавить в модуль urls.py пакета конфигурации такой код:

****

Этот код создает маршрут, связывающий шаблонный путь вида static/<njmb к статическому файлу>1 с контроллером-функцией serve о из модуля django.

Contrib.staticfiles.views, который занимается обработкой запросов к статическим файлам.

У контроллера serve () указан декоратор never cache (), запрещающий кэширование на стороне клиента. Созданный таким образом маршрут добавляется в список маршрутов уровня проекта.

Отметим, что завершающий прямой слеш в шаблонном пути этого маршрута не ставится. Если его все же поставить, то интернет-адреса файлов, ссылки на которые присутствуют в загружаемых таблицах стилей (например, фоновых изображений), будут сформированы некорректно, и эти файлы не загрузятся.

Внимание!

Здесь предполагается, что в качестве префикса, добавляемого к интернет-адресу статического файла, используется значение по умолчанию: ’/static/’ (этот префикс указывается в параметре static url настроек проекта). Если в настройках проекта задан другой префикс, шаблонный путь нужно исправить соответствующим образом.

Если сайт работает в эксплуатационном режиме, функция serve () возбуждает исключение Http404.

□ обязательно очистить кэш веб-обозревателя. Если этого не сделать, веб-обозреватель продолжит использовать копии статических файлов, сохраненные в своем кэше;

□ запустить отладочный веб-сервер с ключом —nostatic, отменяющим обработку статических файлов по умолчанию:



Как только работа над таблицами стилей сайта будет закончена, рекомендуется вновь включить кэширование статических файлов. Для этого достаточно закомментировать строки, добавленные в модуль urls.py пакета конфигурации на первом шаге, и перезапустить отладочный веб-сервер уже без ключа —nostatic.