**Тема занятия № 31: Посредники и обработчики контекста**

1. Посредники

Посредник (middleware) Django — это программный модуль, выполняющий предварительную обработку клиентского запроса перед передачей его контроллеру и окончательную обработку ответа, выданного контроллером, перед отправкой его клиенту. Список посредников, зарегистрированных в проекте, указывается в параметре middleware настроек проекта.

Посредники в Django можно использовать не только для обработки запросов и ответов, но и для добавления в контекст шаблона каких-либо значений. Ту же самую задачу выполняют и обработчики контекста, список которых указывается в дополнительном параметре context processors настроек шаблонизатора.

Посредники — весьма мощный инструмент по обработке данных, пересылаемых по сети. Немалая часть функциональности Django реализована именно в посредниках.

**Стандартные посредники**

Посредники, изначально включенные в список параметра middleware настроек проекта, были описаны ранее. Помимо них, в составе Django имеется еще ряд посредников:

□ django.middleware. Gzip.gzipmiddieware —сжимает запрашиваемую страницу с применением алгоритма gzip, если размер страницы превышает 200 байтов, страница не была сжата на уровне контроллера (для чего достаточно указать у него декоратор gzip pageo). А веб-обозреватель способен обрабатывать сжатые страницы.

В списке зарегистрированных посредников должен находиться перед теми, которые получают доступ к содержимому ответа с целью прочитать или изменить его, и после посредника django.middleware.cache.updatecachemiddleware;

□ django.middleware.http.conditionalgetmiddleware— выполняет обработку заголовков, связанных с кэшированием страниц на уровне клиента. Если ответ не имеет заголовка Е-Tag, такой заголовок будет добавлен. Если ответ имеет заголовки Е-Tag ИЛИ Last-Modified, а Запрос— заголовки If-None-Match ИЛИ ifmodified-since, то вместо страницы будет отправлен ’’пустой” ответ с кодом 304 (запрашиваемая страница не была изменена).

В списке зарегистрированных посредников должен находиться перед Django.middleware. Common.Commonmiddleware;

□ django.middleware.cache.updatecachemiddleware— обновляет КЭШ при включенном режиме кэширования всего сайта.

В списке зарегистрированных посредников должен находиться перед теми, которые модифицируют заголовок Vary (django.contrib.sessions.middleware.

Sessionmiddleware И django.middleware.gzip.gzipmiddleware);

□ django.middleware, cache. Fetchfromcachemiddleware — извлекает запрошенную страницу из кэша при включенном режиме кэширования всего сайта.

В списке зарегистрированных посредников должен находиться после тех, которые модифицируют заголовок Vary (django. Contrib. Sessions.middleware.

Sessionmiddleware И django.middleware.gzip.gzipmiddleware).

Любой из этих посредников в случае необходимости можно вставить в список параметра middleware настроек проекта согласно указаниям касательно очередности их следования.

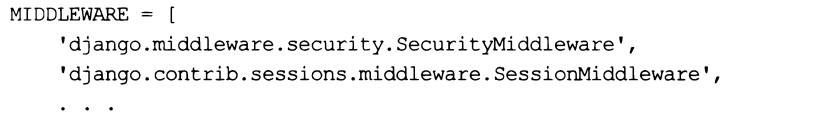
**Порядок выполнения посредников**

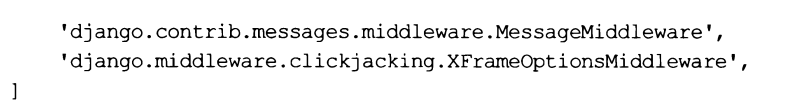
Посредники, зарегистрированные в проекте, при получении запроса и формировании ответа выполняются дважды.

1. Первый раз — при получении запроса, перед передачей его контроллеру, в том порядке, в котором записаны в списке параметра middleware настроек проекта.

2. Второй раз— после того, как контроллер сгенерирует ответ, до отправки его клиенту. Если ответ представлен экземпляром класса templateresponse, то посредники выполняются до непосредственного рендеринга шаблона (что позволяет изменить некоторые параметры запроса — например, добавить какие-либо данные в контекст шаблона). Порядок выполнения посредников на этот раз противоположен тому, в каком они записаны в списке параметра middleware.

Рассмотрим для примера посредники, зарегистрированные во вновь созданном проекте:





При получении запроса сначала будет выполнен самый первый в списке посредник Django.middleware.security.securitymiddleware, далее — django.contrib.sessions, Middleware.sessionmiddleware и т. д. после выполнения посредника django.middleware, Clickjacking. Xframeoptionsmiddleware, последнего в списке, управление будет передано контроллеру.

После того как контроллер сгенерирует ответ, выполнится последний в списке посредник django.middleware.clickjacking.xframeoptionsmiddleware, за НИМ — Предпоследний django.contrib.messages.middleware.messagemiddleware и т. д. После выполнения самого первого В списке посредника django.middleware.security.

Securitymiddleware ответ отправится клиенту.

**Написание своих посредников**

Если разработчику не хватает стандартных посредников, он может написать свой собственный, реализовав его в виде функции или класса.

**Посредники-функции**

Посредники-функции проще в написании, но предоставляют не очень много функциональных возможностей.

Посредник-функция должен принимать один параметр. С ним будет передан либо следующий в списке посредник (если это не последний посредник в списке), либо контроллер (если текущий посредник — последний в списке).

Посредник-функция в качестве результата должна возвращать функцию, в качестве единственного параметра принимающую запрос в виде экземпляра класса httprequest.

В этой ’’внутренней” функции и будет выполняться предварительная обработка запроса и окончательная — ответа в следующей последовательности:

□ если нужно— выполняется предварительная обработка запроса, получаемого возвращаемой функцией в единственном параметре (здесь можно, например, изменить содержимое запроса и добавить в него новые атрибуты).

Внимание!

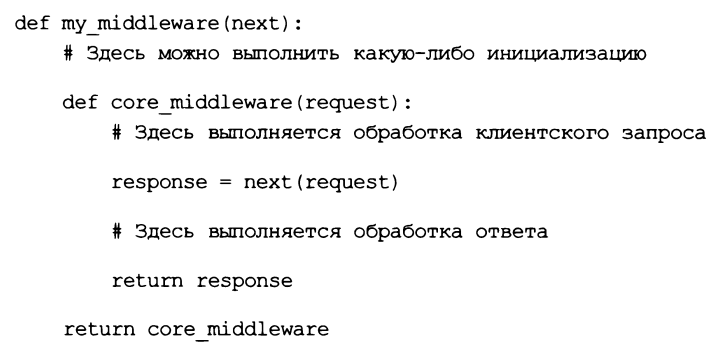
Содержимое можно изменить только у обычного, непотокового ответа. Объект потокового ответа не поддерживает атрибут content, поэтому добраться до его содержимого невозможно.

□ обязательно— вызывается функция, полученная с параметром посредником функцией. В качестве единственного параметра полученной функции передается объект запроса, а в качестве результата она вернет ответ в виде экземпляра класса httpresponse;

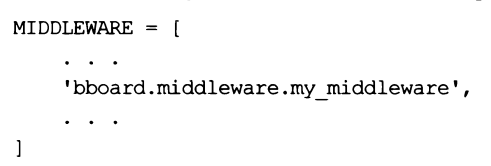
□ если нужно — выполняется окончательная обработка ответа, ранее возвращенного полученной функцией;

□ обязательно — объект ответа возвращается из "внутренней" функции в качестве результата.

Вот своеобразный шаблон, согласно которому пишутся посредники-функции:

****

Регистрируется такой посредник следующим образом (подразумевается, что он объявлен в модуле middleware.py пакета приложения bboard):

****

**Посредники-классы**

Посредники-классы предлагают больше функциональных возможностей, но писать их несколько сложнее.

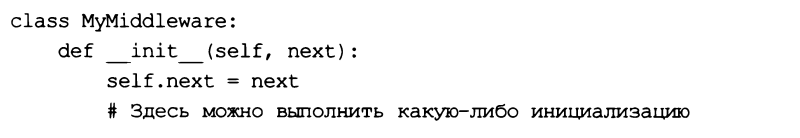
Посредник-класс должен объявлять, по меньшей мере, два метода:

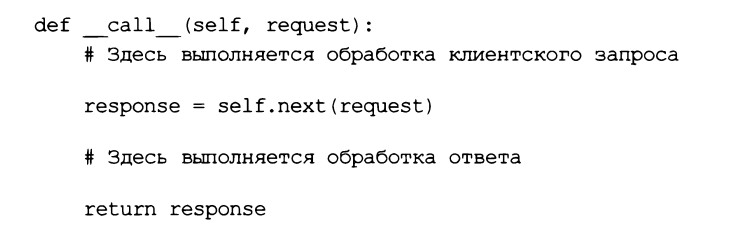
□ конструктор \_\_init\_\_(self, next) — должен принять в параметре next либо следующий в списке посредник, либо контроллер (если посредников больше нет) и сохранить его. Также может выполнить какую-либо инициализацию.

Если в теле конструктора возбудить исключение middiewarenotused из модуля Django.core.exceptions, то посредник деактивируется и более не будет использоваться в дальнейшем;

О \_\_call\_\_(seif, request) — должен принимать в параметре request объект запроса и возвращать объект ответа. Тело этого метода пишется по тем же правилам, что и тело "внутренней" функции у посредника-функции.

Далее приведен аналогичный шаблон исходного кода, согласно которому пишутся посредники-классы:





Дополнительно в посреднике-классе можно объявить следующие методы:

□ process\_view (self, request, view\_func, view\_args, view\_kwargs) — выполняется непосредственно перед вызовом следующего в списке посредника или контроллера (если это последний посредник в списке).

Параметром request методу передается запрос в виде экземпляра класса Httprequest, параметром view func — ссылка на функцию, реализующую контроллер. Это может быть контроллер-функция или функция, возвращенная методом as viewo контроллера-класса. Параметром view args методу передается список позиционных URL-параметров (в текущих версиях Django не используется), а параметром view kwargs — словарь с именованными URL-параметрами, передаваемыми контроллеру.

Метод должен возвращать один из перечисленных далее результатов:

• None— тогда обработка запроса продолжится: будет вызван следующий в списке посредник или контроллер;

• экземпляр класса httpresponse (т. Е. Ответ) — тогда обработка запроса прервется, и возвращенный методом ответ будет отправлен клиенту;

□ process\_exception(self, request, exception) — вызывается при возбуждении исключения в теле контроллера. Параметром request методу передается запрос в виде экземпляра класса httprequest, параметром exception— само исключение в виде экземпляра класса Exception.

Метод должен возвращать:

• None — тогда будет выполнена обработка исключения по умолчанию;

• экземпляр класса httpresponse (т. Е. Ответ) — тогда возвращенный методом ответ будет отправлен клиенту;

О process\_template\_response (self, request, response) — вызывается уже после того, как контроллер сгенерировал ответ, но перед рендерингом шаблона. Параметром request методу передается запрос в виде экземпляра класса httprequest, параметром response — ответ В виде экземпляра класса templateresponse.

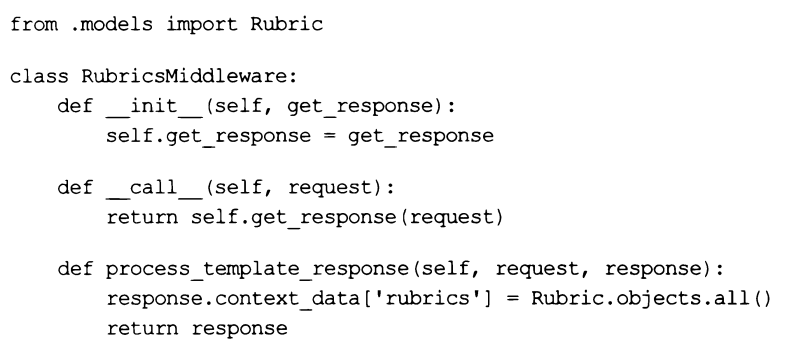
Метод должен возвращать ответ в виде экземпляра класса templateresponse — либо полученный с параметром response и измененный, либо новый, сгенерированный на основе полученного. Этот ответ и будет отправлен клиенту.

Метод может заменить имя шаблона, занеся его в атрибут template name ответа, или содержимое контекста шаблона, доступного из атрибута context data.

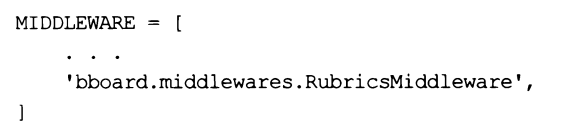
Внимание!

Эффект от замены имени шаблона или изменения содержимого контекста в методе Process template responseo будет ДОСТИГНУТ ТОЛЬКО В ТОМ случае, если ответ представлен экземпляром класса templateresponse.

Приведен код посредника rubricsmiddieware, который добавляет в контекст шаблона СПИСОК рубрик, ВЗЯТЫЙ ИЗ модели Rubric.



He забудем зарегистрировать этот посредник в проекте (предполагается, что он сохранен в модуле bboard.middlewares):

****

После этого мы можем удалить из контроллеров код, добавляющий в контекст шаблона список рубрик, разумеется, при условии, что ответ во всех этих контроллерах формируется В виде экземпляра класса templateresponse.

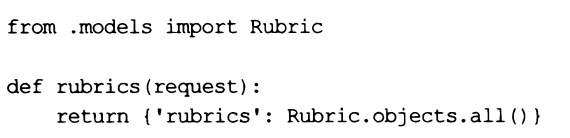
1. Обработчики контекста

Обработчик контекста— это программный модуль, добавляющий в контекст шаблона какие-либо дополнительные данные уже после формирования ответа контроллером.

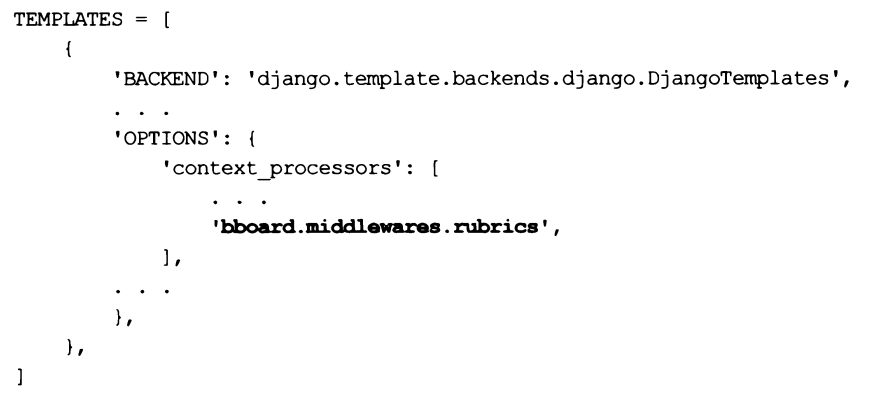
Обработчики контекста удобно использовать, если нужно просто добавить в контекст шаблона какие-либо данные. Обработчики контекста реализуются проще, чем посредники, и работают в любом случае, независимо от того, представлен ответ экземпляром класса templateresponse ИЛИ httpresponse.

Обработчик контекста реализуется в виде обычной функции. Единственным параметром она должна принимать запрос в виде экземпляра класса httprequest и возвращать словарь с данными, которые нужно добавить в контекст шаблона.

Приведен код обработчика контекста rubrics, который добавляет в контекст шаблона список рубрик.

****

Этот обработчик шаблона мы занесем в список параметра context processors, который входит в состав дополнительных параметров используемого нами шаблонизатора:

****