10

Pastebin

Самым привлекательным в платформе Django для многих разработчиков является то, что она позволяет писать веб-приложения на языке Python. Но, как это ни парадоксально, другой привлекательной чертой Django являются ее универсальные представления, дающие возможность создавать веб-приложения, написав совсем немного программного кода на языке Python. (С универсальными представлениями вы познакомились в главе 5 «Адреса URL, механизмы HTTP представления»).

Универсальные представления могут быть мощным инструментом быстрой разработки и макетирования. Но пусть вас не вводит в заблуждение такое их название. Универсальные представления – это не просто временные «подпорки». Начинающим пользователям Django можно простить то, что они видят за этим названием предоставление некоторого шаблона по умолчанию, используемого для отображения данных, но ведь на самом деле реалии совсем не таковы. Не забывайте, что **представление** в платформе Django – это программный код на языке Python, который принимает объект `HttpRequest` запроса HTTP и возвращает объект `HttpResponse` ответа. Разработка структуры объектов данных, передаваемых представлению, и шаблона, используемого для отображения ответа, полностью находятся в вашем ведении.

В этом разделе мы пройдем через процесс создания простого приложения Pastebin[1], которое опирается на использование универсальных

[1] Из Википедии: «pastebin, или nopaste, – веб-приложение, которое позволяет загружать отрывки текста, обычно фрагменты исходного кода, для возможности просмотра окружающими. Такой сервис очень популярен среди пользователей IRC сетей, где вставка больших фрагментов текста считается плохим тоном». – *Прим. перев.*

представлений. Ниже приводится перечень особенностей нашего приложения:

- Форма для передачи новых элементов с одним обязательным (содержимое) и двумя необязательными полями (имя отправителя и заголовок)
- Список последних элементов, щелчком на которых можно выполнять переход к просмотру этих элементов
- Представление для просмотра каждого элемента
- Интерфейс администратора, позволяющий нам (как владельцам сайта) редактировать или удалять существующие записи
- Подсветка синтаксиса
- Периодическое удаление устаревших записей

Для достижения описанных выше функциональных возможностей нам практически не придется писать программный код на языке Python, реализующий управляющую логику. Мы создадим файл модели, описывающей наши данные и их атрибуты и шаблоны отображения этих данных, но практически все остальные аспекты приложения будут реализованы за счет использования встроенных механизмов платформы Django.

Примечание

Программный код этого приложения по своей функциональности напоминает первую версию сайта *dpaste.com* – приложения Pastebin сообщества Django. В настоящее время *dpaste.com* уже не является веб-приложением, опирающимся исключительно на универсальные представления, но сущность его простоты и практичности присутствует в программном коде этой главы.

Одно небольшое примечание, прежде чем мы двинемся дальше: этот пример наглядно демонстрирует, какой значительный объем работы мы можем передать платформе. Некоторые могли бы назвать такой подход подходом для ленивых, но каждая строка программного кода, которую вы не напишете, не потребует отладки в дальнейшем. Поэтому в данном примере везде, где будет возможность выбирать между написанием небольшого объема программного кода, чтобы реализовать некоторые специфические особенности поведения, и возможностью перепоручить работу платформе Django, мы позволим платформе самой выполнять работу.

Определение модели

Ниже приводится полное содержимое файла `models.py` для нашего приложения Pastebin. В нем определяется простая структура данных, состоящая из пяти полей, некоторый параметр в классе `Meta` и пара методов, используемых программным кодом представления и нашими

шаблонами. В нем также выполняется регистрация модели в приложении администрирования и устанавливаются некоторые параметры администрирования, имеющие отношение к отображению списка.

```
import datetime
from django.db import models
from django.db.models import permalink
from django.contrib import admin

class Paste(models.Model):
    """Структура единственной записи в приложении Pastebin"""

    SYNTAX_CHOICES = (
        (0, "Plain"),
        (1, "Python"),
        (2, "HTML"),
        (3, "SQL"),
        (4, "Javascript"),
        (5, "CSS"),
    )

    content = models.TextField()
    title = models.CharField(blank=True, max_length=30)
    syntax = models.IntegerField(max_length=30, choices=SYNTAX_CHOICES,
                                 default=0)
    poster = models.CharField(blank=True, max_length=30)
    timestamp = models.DateTimeField(default=datetime.datetime.now,
                                     blank=True)

    class Meta:
        ordering = ('-timestamp',)

    def __unicode__(self):
        return "%s (%s)" % (self.title or "#%s" % self.id,
                            ➥self.get_syntax_display())

    @permalink
    def get_absolute_url(self):
        return ('django.views.generic.list_detail.object_detail',
            None, {'object_id': self.id})

class PasteAdmin(admin.ModelAdmin):
    list_display = ('__unicode__', 'title', 'poster', 'syntax', 'timestamp')
    list_filter = ('timestamp', 'syntax')

admin.site.register(Paste, PasteAdmin)
```

Большинство этих элементов мы уже видели ранее. Отличие здесь только в строках, содержащих наш собственный программный код на языке Python для данного приложения. За исключением нескольких простых правил в файле `urls.py` основная работа передана самой платформе.

Приложения, опирающиеся на универсальные приложения, такие как в данном примере, на практике демонстрируют мощь принципа

«не повторяйся». В примере приложения, которое мы собираемся создавать, пять полей, составляющих основу описанной выше модели (`content`, `title`, `syntax`, `poster` и `timestamp`), будут использоваться:

- Командой `manage.py syncdb` для создания таблицы в базе данных
- Приложением администрирования для воссоздания интерфейса редактирования наших данных
- Универсальным представлением `object_detail`, которое будет извлекать экземпляры модели и передавать их системе шаблонов для отображения
- Универсальным представлением `create_update`, которое генерирует и обрабатывает форму добавления нового элемента

Методы модели также будут использоваться в нескольких местах. Приложение администрирования будет использовать метод `__unicode__` объекта, когда потребуется сослаться на объект по имени (например, в сообщениях, требующих подтверждения выполнения операции удаления), и метод `get_absolute_url` – для создания ссылок на отдельные записи. Шаблоны будут неявно использовать метод `__unicode__` везде, где потребуется отобразить название элемента, а метод `get_absolute_url` – для создания ссылок в списке последних поступлений.

Создание шаблонов

Теперь создадим несколько простых шаблонов, которые будут использоваться для отображения содержимого сайта. Сначала создадим базовый шаблон, этот прием вы уже видели в главе 7 «Фотогалерея». Сохраните следующие строки в файле `pastebin/templates/base.html`.

```
<html>
    <head>
        <title>{% block title %}{% endblock %}</title>
        <style type="text/css">
            body { margin: 30px; font-family: sans-serif; background: #fff; }
            h1 { background: #ccf; padding: 20px; }
            pre { padding: 20px; background: #ddd; }
        </style>
    </head>
    <body>
        <p><a href="/paste/add/">Add one</a> &bull;
        <a href="/paste/">List all</a></p>
        {% block content %}{% endblock %}
    </body>
</html>
```

После создания базового шаблона перейдем к созданию формы, с помощью которой пользователи смогут добавлять свой программный код в наше приложение. Сохраните следующие строки в файле `pastebin/`

templates/pastebin/paste_form.html. (Кажущаяся избыточность в пути к файлу объясняется ниже.)

```
{% extends "base.html" %}
{% block title %}Add{% endblock %}
{% block content %}
<h1>Paste something</h1>
<form action="" method="POST">
Title: {{ form.title }}<br>
Poster: {{ form.poster }}<br>
Syntax: {{ form.syntax }}<br>
{{ form.content }}<br>
<input type="submit" name="submit" value="Paste" id="submit">
</form>
{% endblock %}
```

Затем создадим шаблон списка. В этом списке будут отображаться последние добавленные элементы, и пользователям будет предоставляться возможность переходить к ним щелчком мыши. Сохраните следующие строки в файле pastebin/templates/pastebin/paste_list.html.

```
{% extends "base.html" %}
{% block title %}Recently Pasted{% endblock %}
{% block content %}
<h1>Recently Pasted</h1>
<ul>
    {% for object in object_list %}
    <li><a href="{{ object.get_absolute_url }}">{{ object }}</a></li>
    {% endfor %}
</ul>
{% endblock %}
```

В заключение создадим шаблон просмотра одного элемента. Это шаблон, на просмотр которого люди будут тратить больше всего времени. Сохраните следующие строки в файле pastebin/templates/pastebin/paste_detail.html.

```
{% extends "base.html" %}
{% block title %}{{ object }}{% endblock %}
{% block content %}
<h1>{{ object }}</h1>
<p>Syntax: {{ object.get_syntax_display }}<br>
Date: {{ object.timestamp|date: "r" }}</p>
<code><pre>{{ object.content }}</pre></code>
{% endblock %}
```

Определение адресов URL

Структура нашего приложения настолько прозрачна, что определения адресов URL выглядят чрезвычайно просто. Единственная хитрость заключается в том, чтобы задействовать универсальные представле-

ния. Нам необходимо определить три шаблона адресов URL: один – для списка всех элементов, один – для отображения отдельных элементов и один – для страницы добавления нового элемента.

```
from django.conf.urls.defaults import *
from django.views.generic.list_detail import object_list, object_detail
from django.views.generic.create_update import create_object
from pastebin.models import Paste

display_info = {'queryset': Paste.objects.all()}
create_info = {'model': Paste}

urlpatterns = patterns('',
    url(r'^$', object_list, dict(display_info, allow_empty=True)),
    url(r'^(?P<object_id>\d+)/$', object_detail, display_info),
    url(r'^add/$', create_object, create_info),
)
```

В действительности эти строки составляют основу нашего приложения. Здесь из пакета `django.view.generic` импортируются три функции представлений:

- `django.views.generic.list_detail.object_list`
- `django.views.generic.list_detail.object_detail`
- `django.views.generic.create_update.create_object`

В дополнение к объекту `HttpRequest`, который все представления Django, как универсальные, так и любые другие, принимают в первом аргументе, этим представлениям также передаются словари с дополнительными значениями. В файле URLconf выше мы определили два различных словаря. Имена `display_info` и `create_info` были выбраны произвольно (хотя суффикс `_info` часто употребляется в именах таких словарей), но их содержимое структурировано с учетом особенностей используемых универсальных представлений. Представление `list_detail` ожидает получить в словаре ключ `queryset`, значением которого является объект `QuerySet`, содержащий все элементы, удовлетворяющие критериям. Представления из модуля `create_update` ожидают получить класс (а не экземпляр) модели в виде значения ключа `model`. Для представления `object_list` мы добавили в словарь ключ `allow_empty=True`, чтобы указать представлению, что страница должна отображаться, даже если в базе данных нет ни одного объекта.

Существует еще множество других значений, которые можно включить в эти словари. Так как они представляют собой основное средство настройки поведения универсальных представлений, они могут иметь довольно существенный объем. Полный перечень параметров, принимаемых этими представлениями, вы найдете в официальной документации к платформе Django. А пока постараемся сохранить максимальную простоту.

Примечание

Существует специальное правило, касающееся использования словарей `_info`, которое вам следует знать. Программный код в файле URLconf не интерпретируется заново при каждом запросе. Это означает, что если не предпринять специальных действий, информация, хранящаяся в объекте `Paste.objects.all`, фактически может устареть по мере добавления, редактирования или удаления записей пользователями или администраторами сайта. К счастью, платформа Django знает об этом и потому ей явно предписывается не кэшировать данные, поставляемые в виде значения ключа `queryset`.

Запуск приложения

Хотя мы написали совсем немного программного кода, у нас теперь имеется достаточно функциональное приложение. Давайте опробуем его. После запуска приложения мы увидели страницу, как показано на рис. 10.1 – пустой список записей.

Отвлечемся на минуту и представим, что потребовалось сделать платформе Django, чтобы получить эту страницу: она проанализировала запрошенный адрес URL; определила, какое представление требуется вызвать; передала (пустой) объект `QuerySet`, полученный из нашей модели; отыскала файл шаблона; отобразила шаблон, используя соответствующее содержимое; и вернула сгенерированную страницу с разметкой HTML броузеру.

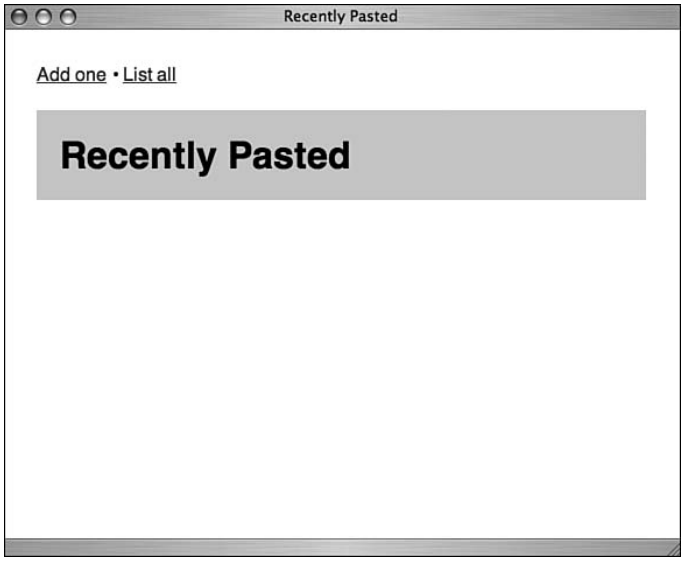

Рис. 10.1. *Пустой список записей*

Теперь добавим некоторое содержимое. Если щелкнуть на ссылке Add One (добавить элемент), в окне броузера должна появиться пустая форма. Эта форма является результатом сотрудничества функции представления Django и нашего шаблона. Универсальное представление `create_update` исследует структуру модели, генерирует элементы формы HTML и передает их нашему шаблону в виде переменной шаблона `{{ form }}`. Наш шаблон распаковывает полученные элементы и отображает форму в окне броузера. Обратите внимание, что ответственность за добавление тега `<form>` и кнопки отправки формы целиком возлагается на нас.

В окончательном виде форма должна выглядеть, как показано на рис. 10.2.

Наше дружественное приложение Pastebin не перегружает пользователя работой. Фактически достаточно будет заполнить лишь поле Code (код). Такие приложения, как Pastebin, будут приносить практическую пользу, только если они удобны в обращении, а длинные списки полей, обязательных к заполнению, – это очень неудобно.

Заполнив форму и щелкнув на кнопке Paste (вставить), мы снова вызываем универсальное представление `create_update` платформы Django. Так как теперь данные передаются методом POST, а не GET, представление понимает, что вместо отображения пустой формы необходимо обработать ввод пользователя и (если это возможно) сохранить его в базе данных.

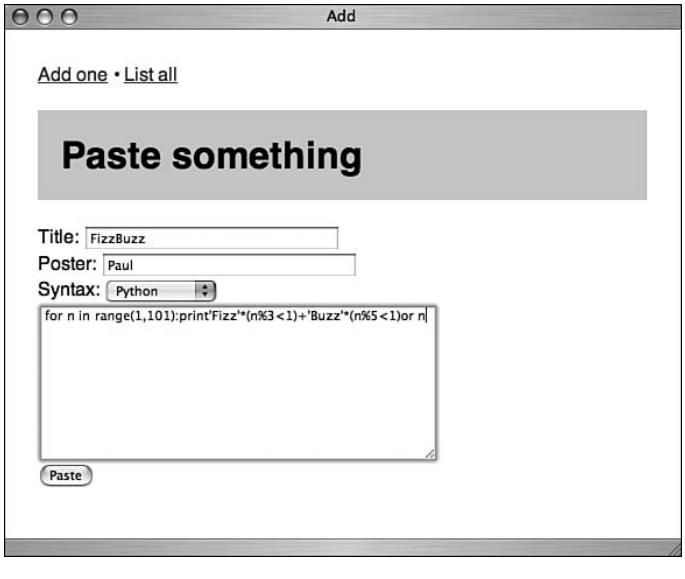

Рис. 10.2. *Форма добавления одной записи*

Одной из особенностей представления `create_update`, которую мы не рассматривали здесь, является механизм проверки корректности введенных данных. Что произойдет, если пользователь оставит обязательное поле незаполненным? В этом случае форма будет выведена повторно. В нашем чрезвычайно простом примере отсутствует программный код, который принимал бы и отображал сообщения об ошибках, но в действительности платформа Django *передает* их в виде переменной шаблона `{{ form.errors }}`. Более надежная реализация могла бы принимать эти сообщения об ошибках и отображать их для обеспечения большего удобства.

Предположим, что пользователь заполнил единственное обязательное поле и щелкнул на кнопке Paste (вставить), а платформа Django обработала введенные данные (с помощью все того же универсального представления `create_update`) и сохранила их в базе. После этого она перенаправит пользователя к вновь созданному объекту, по адресу `get_absolute_url`, и отобразит отправленные данные с помощью шаблона `pastebin_detail.html`, как показано на рис. 10.3.

Наиболее ожидаемое поведение, на которое рассчитывает пользователь после отправки новой записи, – представление этой записи и адрес URL, который он сможет отправить другим.

Если пользователь пожелает узнать, какие еще записи присутствуют в приложении Pastebin, ссылка List All (вывести все) даст ему такую возможность. Все имеющиеся записи будут выводиться в виде просто-

Рис. 10.3. *Вновь добавленная запись*

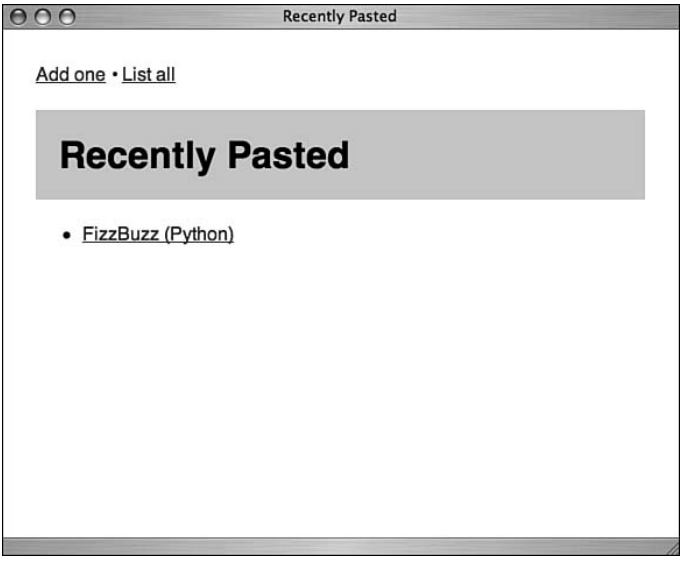

Рис. 10.4. *Список отправленных записей*

го маркированного списка, как показано на рис. 10.4, с помощью универсального представления `object_list` и шаблона `paste_list.html`.

Этот список прекрасно иллюстрирует работу универсального представления `object_list`. С помощью словаря `display_info`, который определен в нашем файле URLconf, в представление `object_list` передается объект `QuerySet` со всеми нашими объектами `Paste`. Это представление передает объекты нашему шаблону, где цикл `{% for object in object_list %}` превращает их в ссылки.

Примечание

С практической точки зрения, список всех записей необязательно является самой значимой особенностью сайтов данного типа. Пользователи приложения Pastebin обычно проявляют интерес только к своим записям. Владельцы приложений типа Pastebin часто задаются вопросом: «Почему спамеры все время что-то добавляют в мое приложение?». Ответ прост: спамеров интересуют любые механизмы, позволяющие показать свою информацию ничего подобного не ожидающим пользователям. Список последних поступлений в Pastebin прекрасно справляется с поставленной задачей, при этом оставаясь неудобным для размещения коммерческих объявлений.

Наконец, не забывайте, что сверх всего того, что мы создали более или менее явно, мы получаем еще и приложение администрирования. Учитывая параметры, которые были определены в классе `PasteAdmin`, административный раздел нашего сайта выглядит, как показано на рис. 10.5.

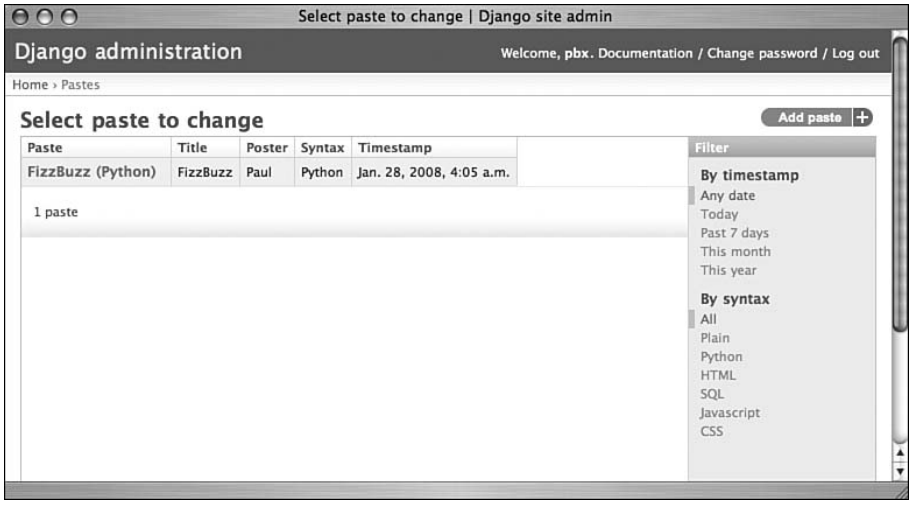

Рис. 10.5. *Страница администратора*

Обратите внимание, что первым аргументом методу `list_display` класса `PasteAdmin` передается не какое-то определенное поле модели, а метод `__unicode__`, благодаря чему создаются ссылки, текст которых зависит от того, какая информация доступна.

Это полезная возможность для приложений, полностью опирающихся на использование универсальных представлений платформы Django. Хотя было бы вполне разумно начинать расширение возможностей приложения с создания специализированных функций представлений, тем не менее, множество улучшений можно внести, продолжая использовать универсальные представления. Мы внесем три улучшения: добавим возможность управления списком последних поступлений, подсветку синтаксиса и автоматическое удаление устаревших записей.

Ограничение числа записей в списке последних поступлений

Список последних поступлений выглядит прекрасно при небольшом количестве записей, но если сайт имеет высокую посещаемость, такой список может очень быстро стать слишком громоздким. Существуют разные способы ограничить число записей в этом списке. Учитывая, что мы используем только универсальные представления, лучшим местом, где можно было бы реализовать эти ограничения, является шаблон.

Для этого достаточно изменить шестую строку шаблона `paste_list.html`, применив фильтр `slice` к объекту `object_list`.

```
{% for object in object_list|slice:":10" %}
```

Вот как это работает: файл URLconf передает в шаблон объект `QuerySet`, представляющий все записи в базе данных. Они уже сортированы в порядке отдаления времени добавления – благодаря наличию строки `ordering = ('-timestamp')` в файле `models.py`. Цикл `for` шаблона извлекает первые десять записей и выполняет итерации по ним.

Значение, передаваемое фильтру `slice`, в точности соответствует значению, которое мы использовали бы в обычной операции извлечения среза на языке Python, за исключением скобок. Эквивалентный пример на языке Python можно представить примерно так:

```
>>> number_list = [15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
>>> print number_list[:10]
[15, 14, 13, 12, 11, 10, 9, 8, 7, 6]
```

Если бы объекты `QuerySet` не откладывали выполнение операций с базой данных, то есть, если бы шаблону передавался полный список объектов только для того, чтобы выбросить все записи, кроме первых десяти, это было бы безумием. Если бы у нас имелись тысячи записей, потребление памяти процессом веб-сервера взлетело бы до предела. Благодаря тому, что объекты `QuerySet` откладывают выполнение операций с базой данных до самого последнего момента (в данном случае таким последним моментом является цикл `for` в нашем шаблоне), мы без всякого ущерба можем указать в файле URLconf вызов метода `Paste.objects.all()` и затем получать срез в шаблоне. Кроме того, поскольку число, отражающее количество элементов в списке, относится к отображению, шаблон является отличным местом для реализации такого ограничения.

Подсветка синтаксиса

Приложение Pastebin будет намного полезнее (и привлекательнее), если оно будет знать, как применять подсветку синтаксиса к добавляемым фрагментам. Существуют различные способы реализации этой возможности (включая замечательную библиотеку **Pygments** на языке Python, которая используется сайтом *dpaste.com*), но самый простой способ заключается в том, чтобы реализовать подсветку синтаксиса на языке JavaScript на стороне клиента.

Программист Алекс Горбачев (Alex Gorbatchev) создал на языке JavaScript отличную реализацию подсветки синтаксиса. Называется эта утилита Syntax Highlighter, а получить ее можно на сайте Google Code (*http://code.google.com/p/syntaxhighlighter/*).

Полный комплект инструкций и примеров использования Syntax Highlighter можно найти на веб-сайте проекта, а здесь мы лишь покажем, как добавить подсветку синтаксиса для фрагментов программного кода на языке Python.

Сначала дополним шаблон `paste_detail.html`, который теперь выглядит, как показано ниже:

```
{% extends "base.html" %}
{% block title %}{{ object }}{% endblock %}
{% block content %}
<h1>{{ object }}</h1>
<p>Syntax: {{ object.get_syntax_display }}<br>
Date: {{ object.timestamp|date:"r" }}</p>
<code><pre name="code" class="{{ object.get_syntax_display|lower }}">
    {{ object.content }}</pre></code>
<link type="text/css" rel="stylesheet"
    href="/static/css/SyntaxHighlighter.css"></link>
<script language="javascript" src="/static/js/shCore.js"></script>
<script language="javascript" src="/static/js/shBrushPython.js"></script>
<script language="javascript" src="/static/js/shBrushXml.js"></script>
<script language="javascript" src="/static/js/shBrushJscript.js"></script>
<script language="javascript" src="/static/js/shBrushSql.js"></script>
<script language="javascript" src="/static/js/shBrushCss.js"></script>
<script language="javascript">
    dp.SyntaxHighlighter.HighlightAll('code');
</script>
{% endblock %}
```

Мы добавили в тег `<pre>` атрибуты `name` и `class`. Это позволяет программному коду JavaScript отыскивать наши фрагменты.

```
<pre name="code" class="{{ object.get_syntax_display|lower }}">
```

Это все, что потребовалось. Когда броузер будет отображать страницу, он выполнит программный код JavaScript, реализующий подсветку синтаксиса, который преобразит черно-белые невыразительные фрагменты программного кода, прежде чем пользователь увидит их. Результат должен выглядеть примерно так, как показано на рис. 10.6.

Удаление устаревших записей с помощью задания cron

Записи, отправляемые на сайты Pastebin, обычно недолго сохраняют свою актуальность, поэтому будет нелишним организовать периодическую проверку и автоматическое удаление устаревших записей. Лучше всего реализовать такую очистку в виде **задания cron**, запускаемого каждую ночь на вашем сервере.

Задания cron и прочие сценарии Django, которые предназначены для работы вне окружения веб-сервера, – это еще одно свидетельство практической ценности принципа, утверждающего, что Django – «это всего лишь программный код на языке Python». При создании сценариев, имеющих дело с объектами приложений на платформе Django, при-

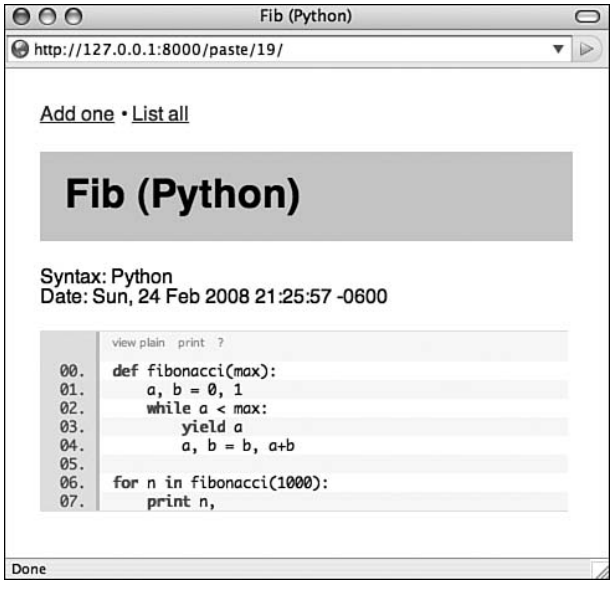

Рис. 10.6. *Фрагмент программного кода на языке Python с подсветкой синтаксиса*

влекается не так много особенностей, специфичных для Django. Наш простой сценарий зависит от следующих предположений:

- Переменная окружения `DJANGO_SETTINGS_MODULE` содержит строку, определяющую на языке Python путь к файлу с настройками проекта (например, «pastesite.settings»).
- В модуле `settings` проекта имеется параметр `EXPIRY_DAYS`.
- Проект называется «pastesite».

Предполагая, что об удалении устаревших записей мы позаботились, останется позаботиться только о тестировании и развертывании приложения.

```
#!/usr/bin/env python
import datetime
from django.conf import settings
from pastesite.pastebin.models import Paste

today = datetime.date.today()
cutoff = (today - datetime.timedelta(days=settings.EXPIRY_DAYS))
Paste.objects.filter(timestamp__lt=cutoff).delete()
```

Основная работа выполняется в последней строке сценария – она использует механизм ORM Django для выбора всех объектов в базе данных, добавленных раньше расчетного времени, и удаляет их все сразу.

Примечание

В зависимости от используемого механизма баз данных, можно также «вычищать», или освобождать пространство, занимаемое удаленными записями. Простейший вариант реализации такой очистки для базы данных SQLite можно найти по адресу: *http://djangosnippets.org*.

В заключение

Хотелось бы надеяться, что теперь вы убедились в широких возможностях универсальных представлений Django. Наш пример сайта Pastebin оказался достаточно прост в реализации, но только представьте все особенности, которыми он обладает: проверка вводимых данных, перенаправление после отправки формы, возможность просмотра списка и отдельных записей и т. д. Но, пожалуй, еще лучше, чем наличие всех этих особенностей, – это знание, что благодаря использованию платформы Django мы имеем надежную основу для внесений расширений. Если нам потребуется внести в наше приложение некоторые ограничения или уменьшить объем внутреннего трафика, добавив кэширование, то платформа Django всегда у нас под рукой.