Скрапинг веб-сайтов

с помощью Python

Р. Митчелл

ДМК ПРЕСС
ИЗДАТЕЛЬСТВО

ИД «Гевисста»

O'REILLY®

Ryan Mitchell

Web Scraping with Python

COLLECTING DATA FROM THE MODERN WEB

Beijing · Boston · Farnham · Sebastopol · Tokyo

O'REILLY®

Райан Митчелл

Скрапинг веб-сайтов с помощью Python

СБОР ДАННЫХ ИЗ СОВРЕМЕННОГО ИНТЕРНЕТА

ДМК ПРЕСС ИЗДАТЕЛЬСТВО
Москва, 2016

ИЦ «Гевисста»

УДК 004.738.1 :004.738.52Python
ББК 32.971.353
М66

Митчелл Р.
М66 Скрапинг веб-сайтов с помощью Python / пер. с англ. А. В. Груздев. – М.: ДМК Пресс, 2016. – 280 с.: ил.

ISBN 978-5-97060-223-2

Изучите методы скрапинга и краулинга веб-сайтов, чтобы получить доступ к неограниченному объему данных в любом уголке Интернета в любом формате. С помощью этого практического руководства вы узнаете, как использовать скрипты Python и веб-API, чтобы одновременно собрать и обработать данные с тысяч или даже миллионов веб-страниц.

Идеально подходящая для программистов, специалистов по безопасности и веб-администраторов, знакомых с языком Python, эта книга знакомит не только с основными принципами работы веб-скраперов, но и углубляется в более сложные темы, такие как анализ сырых данных или использование скраперов для тестирования интерфейса веб-сайта. Примеры программного кода, приведенные в книге, помогут разобраться в этих принципах на практике.

УДК 004.738.1 :004.738.52Python
ББК 32.971.353

ISBN 978-1-491-91029-0 (англ.) © 2015 Ryan Mitchell
ISBN 978-5-97060-223-2 (рус.) © Оформление, перевод, ДМК Пресс, 2016

Содержание

Предисловие

В современном цифровом мире данные приобретают все большую и большую ценность. Если мы посмотрим вокруг, то увидим огромное количество различных сервисов, которые пытаются сделать нашу жизнь лучше. Стараются нам помочь, посоветовать, найти нужную информацию. От гигантов типа Google до стартапов и небольших экспериментальных проектов, все эти сервисы работают с данными. В основе любой задачи, которую предстоит сегодня решать машине или человеку, лежат данные.

С другой стороны, на недостаток данных в современном мире жаловаться не приходиться. Данные с большой скоростью генерируются и накапливаются компаниями и устройствами, объемы хранения растут но, конечно, самым перспективным источником данных является сеть Интернет.

Когда-то Интернет был маленькой американской сетью для нескольких сотен человек, где почти все друг друга знали. Теперь это гигантская информационная структура. Здесь практически невозможно контролировать потоки информации. То, что называется «контент, генерируемый пользователями» составляет колоссальный объем данных. Этот объем уже даже невозможно точно измерить. Точно так же как мы когда-то перестали измерять расстояние до звезд в километрах, применяемых на Земле, и стали использовать понятие «световой год», то есть характеристику скорости, также и об интернет-данных теперь пишут в терминах скорости прироста информации, а не ее объема. Так, за одну минуту в Интернете появляются, например, более 3 миллионов новых постов в сети Facebook, более 300 тысяч сообщений в Twitter, более 30 тысяч отзывов о книгах и покупках, не говоря уже об описаниях новых фильмов, товаров и т. д., и т. п.

Все это многообразие информации сейчас активно используется не только людьми, но и организациями для повышения эффективности своей деятельности. Жизнь людей сейчас настолько быстро меняется, что традиционных анкетных данных просто недостаточно для оценки поведения заемщика (если это банк), покупателя (если это розничная сеть). В стремительно меняющемся мире они банально устаревают. И тогда нужно обратиться к «внешним» данным. Произвести обогащение текстовыми данными. Банкам важно анализировать, что пользователи пишут в социальных сетях, на сайтах. Оценить их уровень грамотности, словарный запас, тематику публикуемых сообщений. Магазинам, производителям важно собирать отзывы покупателей об

их товарах, анализировать их тональность, чтобы лучше спланировать рекламную компанию, использовать в ней текстовые формулировки, максимально близкие покупателю. Мы, в своей работе с различными компаниями, видим большое количество таких примеров и перечислять их можно бесконечно.

Тем не менее, для того чтобы эффективно работать с этой информацией, получать из нее пользу и реализовывать задачи, востребованные компаниями и людьми, данные нужно извлекать, обрабатывать, структурировать. То, что мы видим как веб-сайт с отзывом о фильме, для машины представляется сборищем разных «кусков» данных с непонятным назначением. Человек, взглянув на веб-страницу, сразу легко определяет нужный и значимый раздел, но для компьютера понимание того, какой именно текст следует обрабатывать, как отделить этот текст от рекламы, ненужных заголовков, ссылок является довольно сложной задачей.

По мере роста потока информации, возможностей по применению этой информации в прикладных задачах, развиваются технические подходы, объединяемые общим термином «веб-краулинг» или «веб-скрапинг». Они предназначены для сбора информации из сети Интернет и ее подготовки к автоматизированной обработке. И несмотря на то, что не всегда процесс веб-скрапинга «виден» для конечного пользователя, часто именно он является ключевым моментом современных веб-технологий. Возьмем для примера тот же Google. Все мы пользуемся поиском, все мы восхищаемся качеством и релевантностью выдаваемых результатов, но, перед тем как ответить на наши поисковые запросы, сервисы Google проводят огромную работу, они обходят все сайты, скачивают с них страницы, выполняют синтаксический разбор контента, определяют, какой текст на этих страницах является «значимым» и именно его индексируют и выдают пользователю.

Разработка и реализация качественных механизмов сбора информации является залогом успешной ее обработки и в этой книге дается детальное руководство по подходам и методам решения этой задачи с помощью популярного языка программирования Python.

Сейчас Python – стремительно развивающийся язык. В первую очередь его популярность связана с простотой и легкостью освоения. Также развитию популярности способствует и то, что он поставляется под открытой лицензией и является свободно распространяемым. Это обеспечивает наличие огромного количества библиотек, расширений и готовых компонентов, которые можно свободно использовать

в своих проектах, что сильно экономит время разработчика. Python – язык с большой и богатой историей. Он появился более 25 лет назад как высокоуровневый язык программирования – альтернатива Java и C++, на котором всего в несколько строчек можно описать то, что на этих низкоуровневых языках программирования занимает по несколько блоков кода. Python в первую очередь предназначен для написания прикладных приложений, но за годы существования он развился в очень гибкий инструмент, на котором сейчас уже пишутся и очень большие, серьезные и высоконагруженные проекты.

Дополнительным преимуществом данной книги является не только использование Python, но и форма подачи материала. В книге даны не просто примеры кода, весь материал представлен в виде примеров конкретных и практических задач.

Нельзя не отметить и наличие целого учебного сайта, разработанного Райан. На его примере наглядно показана отправка форм, работа капчи, скрапинг JavaScript (использование пауз в выполнении скриптов) и т. д.

Очень приятно, что данная книга выходит в свет на русском языке. Авторы и переводчики проделали огромную и качественную работу. Надеюсь, что распространение знаний и методов, описанных в данной книге, будет направлено разработчиками на создание качественно новых, интеллектуальных сервисов, которые будут приносить пользу людям и организациям в нашей повседневной жизни и, конечно, сделают ее немного лучше.

Денис Афанасьев,
генеральный директор компании «CleverDATA»

Вступление

Для новичков программирование может показаться своего рода магией. И если программирование – это магия, то *веб-скрапинг (web scraping)* является колдовством. Применение магии для решения конкретных впечатляющих и важных задач, да еще без всяких усилий, выглядит чудом.

В самом деле, когда я работала инженером-программистом, я пришла к выводу, что существует очень немного направлений в программировании, которые так же, как веб-скрапинг, одинаково привлекают внимание программистов и обычных пользователей. Хотя написать простую программу-робот, которая соберет информацию, отправит ее в терминал или сохранит в базе данных, несложно, все равно испытываешь определенный трепет, ощущение непредсказуемости результата, независимо от того, сколько раз ты уже проделывал эту процедуру.

Жаль, что когда я разговариваю с другими программистами о веб-скрапинге, возникает много непонимания и путаницы. Некоторые высказывают сомнения по поводу законности веб-скрапинга или спорят по поводу способов обработки современного Интернета со всеми его JavaScript, мультимедийным контентом и cookies. Некоторые смешивают API с веб-скраперами.

Настоящая книга стремится поставить окончательную точку в этих наиболее распространенных вопросах и заблуждениях, сложившихся вокруг веб-скрапинга, дает развернутое руководство по решению наиболее востребованных задач веб-скрапинга.

Начиная с главы 1, я буду периодически приводить примеры программного кода, чтобы продемонстрировать основные принципы веб-скрапинга. Эти примеры являются общественным достоянием, и их можно свободно использовать (тем не менее соблюдение авторских прав всегда приветствуется). Кроме того, все примеры программного кода будут размещены на веб-сайте для просмотра и скачивания.

Что такое веб-скрапинг?

Автоматизированный сбор данных из Интернета существует столько же, сколько сам Интернет. Несмотря на то что *веб-скрапинг (web scraping)* не является новым термином, раньше это направление было больше известно под названием *анализ экранных или интерфейсных данных (screen scraping)*, *интеллектуальный анализ данных (data mining)*, *сбор веб-данных (web harvesting)*. Похоже, что на сегодняшний

день общее мнение склоняется в пользу термина *веб-скрапинг (web scraping)*, который я и буду использовать на протяжении всей книги, хотя время от времени буду называть программы веб-скрапинга *роботами (bots)*.

В теории веб-скрапинг – это сбор данных с помощью любых средств, кроме программ, использующих API (или человека, использующего веб-браузер). Чаще всего веб-скрапинг осуществляется с помощью программы, которая автоматически запрашивает веб-сервер, запрашивает данные (HTML и другие файлы, которые размещены на веб-страницах), а затем выполняет парсинг этих данных, чтобы извлечь необходимую информацию.

На практике веб-скрапинг охватывает широкий спектр методов и технологий программирования, таких как анализ данных и информационная безопасность. Эта книга посвящена основам веб-скрапинга и краулинга (часть I)[1] и раскрывает некоторые сложные темы (часть II).

Зачем нужен веб-скрапинг?

Если для вас единственным способом доступа к Интернету является браузер, вы теряете огромный спектр возможностей. Хотя браузеры удобны для выполнения JavaScript, вывода изображений и представления объектов в более удобочитаемом формате (помимо прочего), веб-скраперы удобны для сбора и обработки больших объемов данных (помимо прочего). Вместо однократного просмотра одной страницы на дисплее монитора вы можете просматривать базы данных, которые уже содержат тысячи или даже миллионы страниц.

Кроме того, веб-скраперы могут проникнуть в такие места, куда традиционные поисковые системы проникнуть не могут. Поиск Google по «cheapest flights to Boston» выдаст множество рекламных сайтов и популярных сайтов заказа авиабилетов. Google возвращает лишь то, что эти веб-сайты сообщают на своих страницах, а не точные результаты в ответ на различные запросы, введенные в системе зака-

---

[1] Обратите внимание, Райан проводит разницу между терминами *веб-скрапер* (*web-scraper*) и *веб-краулер* (*web-crawler*). Веб-скрапер – это программа, которая собирает данные и извлекает нужную информацию с одной или нескольких страниц сайта. Веб-краулер – это программа, которая просто обходит или сканирует страницы на одном или нескольких сайтах. В этом плане веб-краулер является синонимом термина *поисковый робот* (*web-spider*). – *Прим. пер.*

за авиабилетов. Тем не менее правильно разработанный веб-скрапер может собрать данные о ценах на авиабилеты до Бостона за определенный временной интервал на различных веб-сайтах и подсказать оптимальное время для покупки авиабилета.

Вы, вероятно, спросите: «Разве сбор данных – это не то, для чего используется API?» (Если вы не знакомы с API, обратитесь к главе 4.) Ну, тогда API – это фантастическое средство, если вы с его помощью сразу найдете то, что искали. Они могут обеспечить передачу потока данных определенного формата с одного сервера на другой. вы можете использовать API для получения различных видов интересующих вас данных, например, твитов или страниц Википедии. В общем случае лучше использовать API (если он есть), а не создавать программу-робот для сбора тех же самых данных. Тем не менее есть несколько причин, в силу которых использовать API не представляется возможным:

- вы собираете данные с сайтов, которые не имеют единого API;
- данные, которые вас интересуют, компактны, поэтому API не нужен;
- источник данных не имеет инфраструктуры или технических возможностей, позволяющих разработать API.

Даже в тех случаях, когда API *уже есть*, объем обрабатываемых запросов, ограничения скорости обработки запросов, тип или формат возвращаемых данных могут не удовлетворить ваши потребности.

Вот именно здесь и начинается сфера применения скрапинга веб-страниц. За некоторыми исключениями, вы можете просмотреть эти страницы в браузере или получить доступ к ним с помощью скрипта Python. Получив доступ к ним с помощью скрипта, вы можете сохранить их в базе данных. Сохранив их в базе данных, вы можете выполнять любые действия с ними.

Очевидно, что существует очень много практических сфер, где требуется доступ к данным практически неограниченного объема. Рыночное прогнозирование рынка, машинный перевод и даже медицинская диагностика уже извлекли огромную пользу, воспользовавшись возможностью собрать и проанализировать данные новостных сайтов, переведенный контент и сообщения на медицинских форумах.

Даже в мире искусства веб-скрапинг уже открыл новые горизонты для творчества. В рамках проекта 2006 года «We Feel Fine» (http://wefeelfine.org/) Джонатан Харрис и Сеп Камвар провели скрапинг англоязычных блогов для поиска фраз, начинающихся с «I feel» или «I am

feeling». Это позволило построить визуализацию данных, описать, как люди в мире чувствуют себя изо дня на день, с минуты на минуту.

Независимо от вашей предметной области, почти всегда есть способ, благодаря которому веб-скрапинг может повысить эффективность бизнес-практик, улучшить производительность или даже открыть совершенно новое направление в бизнесе.

Об этой книге

Эту книгу можно рассматривать не только как введение в веб-скрапинг, но и как развернутое руководство по скрапингу веб-данных практически любого типа. Хотя в книге используется язык программирования Python и освещаются основные принципы его работы, ее не следует использовать в качестве вводного пособия по Python.

Если вы не являетесь опытным программистом и не знаете Python вообще, чтение этой книги может быть несколько сложной задачей. Однако если вы опытный программист, то сочтете материал книги легким. В приложении А освещаются установка и работа с Python 3.x, который используется в этой книге. Если вы работали только с Python 2.x или у вас не установлен Python 3.x, вы, возможно, захотите ознакомиться с приложением А.

Если вы ищете более подробные ресурсы по изучению Python, книга *Introducing Python* от Билла Любановича является очень хорошим развернутым руководством. Для тех, у кого мало времени, видеокурс *Intoduction to Python* от Джессики МакКеллер является отличным ресурсом.

Приложение С включает в себя кейсы, а также в нем рассматриваются ключевые вопросы, касающиеся правовых аспектов использования скраперов в США и данных, полученных с их помощью.

В технической литературе внимание часто уделяется конкретному языку или технологии, однако веб-скрапинг является относительно разрозненной предметной областью, которая охватывает такие направления, как работа с с базами данных, веб-серверами, HTTP, HTML, интернет-безопасность, обработка изображений, наука о данных (data science) и др. Эта книга стремится осветить все эти направления в том объеме, в каком это требуется для сбора данных в Интернете.

В первой части подробно рассказывается о веб-скрапинге и веб-краулинге, значительное внимание уделено работе библиотек, использующихся в книге. Первую часть книги можно использовать в качестве развернутого справочного пособия по этим библиотекам

и методам (за некоторыми исключениями, когда будут приводиться дополнительные источники).

Во второй части освещаются дополнительные темы, которые могут пригодиться читателю при написании веб-скрапера. К сожалению, эти темы являются слишком широкими, чтобы подробно рассмотреть их в одной главе. Поэтому часто будут приводиться ссылки на другие ресурсы для получения дополнительной информации.

Структура данной книги составлена так, что можно легко перемещаться по главам в поисках необходимой информации. Если какой-то принцип или фрагмент программного кода связан с другим принципом или фрагментом, уже упоминавшимся в предыдущей главе, я обязательно сошлюсь на соответствующий раздел.

Типографские соглашения

В этой книге применяются следующие типографские соглашения:

Курсив
: Используется для обозначения новых терминов, URL-адресов, адресов электронной почты, имен файлов и расширений файлов.

`Моноширинный шрифт`
: Используется для листингов программ, а также внутри параграфов для обозначения элементов программ (названий переменных или функций, баз данных, типов данных, переменных среды, операторов и ключевых слов).

`Моноширинный жирный шрифт`
: Обозначает команды или другой текст, который должен вводиться пользователем.

`Моноширинный курсив`
: Обозначает текст, который должен замещаться фактическими значениями, вводимыми пользователем или определяемыми из контекста.


Этот элемент означает совет или подсказку.


Этот элемент означает примечание.


Этот элемент означает предупреждение или предостережение.

Использование примеров программного кода

Все примеры программного кода и упражнения, что приводятся в этой книге, доступны для скачивания по адресу http://pythonscraping.com/code/.

Данная книга призвана оказать вам помощь в решении задач, связанных с веб-скрапингом. Вы можете свободно использовать примеры программного кода из этой книги в своих программах и документации. Вам не нужно обращаться в издательство за разрешением, если вы не собираетесь воспроизводить существенные части программного кода. Например, если вы разрабатываете программу и используете в ней несколько фрагментов программного кода из книги, вам не нужно обращаться за разрешением. Однако в случае продажи или распространения компакт-дисков с примерами из этой книги вам необходимо получить разрешение от издательства O'Reilly. Если вы отвечаете на вопросы, цитируя данную книгу или примеры из нее, получения разрешения не требуется. Но при включении значительного объема программного кода из этой книги в Вашу документацию необходимо получить разрешение издательства.

Мы приветствуем, но не требуем добавлять ссылку на первоисточник при цитировании. Под ссылкой на первоисточник мы подразумеваем указание авторов, издательства и ISBN. Например: *Web Scraping with Python* by Ryan Mitchell (O'Reilly). Copyright 2015 Ryan Mitchell, 978-1-491-91029-0.

Если вы считаете, что использование вами примеров программного кода выходит за разрешенные рамки, присылайте свои вопросы на нашу электронную почту permissions@oreilly.com.

Благодарности

Подобно тому, как наилучшие продукты появляются благодаря постоянной обратной связи с пользователями, так и выход этой книги в той или иной форме был бы невозможен без участия большого числа коллег, вдохновителей и редакторов. Спасибо сотрудникам O'Reilly за всестороннюю поддержку этой несколько необычной темы, а также моим друзьям и семье, которые давали советы и вынуждены были слушать мои импровизированные чтения, моих коллегам из компании LinkeDrive, которой я, вероятно, задолжала уйму рабочего времени.

Я выражаю благодарность, в частности, Эллисону Макдональду (Allyson MacDonald), Брайану Андерсону (Brian Anderson), Мигелю Гринбергу (Miguel Grinberg) и Эрику ВанВику (Eric VanWyk) за обратную связь, рекомендации и иногда жесткую критику из лучших побуждений. Довольно большое количество разделов и примеров программного кода явилось прямым результатом их вдохновляющих предложений.

Я выражаю благодарность Йейлу Шпехту (Yale Specht), изначально сподвигнувшему меня на этот проект и вносившему стилистические поправки по мере написания книги, за его безграничное терпение на протяжении последних девяти месяцев. Без него эта книга была бы написана в два раза быстрее, но не была бы столь полезной.

И наконец, я выражаю благодарность Джиму Вальдо (Jim Waldo), с которого все, в общем-то, и началось, когда он отправил по почте молодому и впечатлительному подростку компьютер с Linux и книгу *The Art and Science of C*.

ЧАСТЬ I

ПОСТРОЕНИЕ СКРАПЕРОВ

В этой части книги основное внимание уделено основным принципам работы веб-скраперов: как использовать Python, чтобы получить информацию с веб-сервера, как выполнить обработку ответа сервера и как начать работать с веб-сайтом в автоматическом режиме. В итоге вы будете с легкостью собирать данные по всему Интернету, создавая скраперы, которые могут переходить с одного домена на другой, собирать информацию и сохранять ее для дальнейшего использования.

Честно говоря, веб-скрапинг – это фантастическое поле деятельности, которое обязательно нужно освоить, если вы хотите получить огромный выигрыш при сравнительно небольших первоначальных вложениях. По всей вероятности, 90% проектов по веб-скрапингу, с которыми вы столкнетесь, будут опираться на методы, используемые лишь в первых шести главах. В этой части освещаются темы, которые чаще всего возникают в головах людей (хотя и технически подкованных) при упоминании слова «веб-скрапер»:

- сбор HTML-данных с домена;
- парсинг данных с целью получения интересующей информации;
- хранение извлеченной информации;
- перемещение на другую страницу для повторения процесса (опционно).

Эта часть книги составит прочную основу ваших знаний, прежде чем вы перейдете к более сложным проектам, рассмотренным во второй части. Не обманывайте себя, думая, что проекты, приведенные в первой части, не столь важны, в отличие от некоторых сложных проектов, рассмотренных во второй части. При разработке скраперов вы будете регулярно использовать информацию, изложенную в первой части книги.

Глава 1

Ваш первый скрапер

Приступая к веб-скрапингу, начинаешь ценить все мелкие настройки, kotcisesopые выполняют браузеры за нас. Веб, лишенный HTML-форматирования, CSS-стилей, JavaScript и рендеринга изображений, может поначалу немного напугать своим видом, но в этой главе, а также в следующей мы расскажем, как форматировать и интерпретировать данные без помощи браузера.

Эта глава сначала расскажет, как отправить GET-запрос к веб-серверу на сканирование конкретной страницы, считав HTML-вывод и выполнив извлечение данных так, чтобы собрать только интересующий нас контент.

Соединение с Интернетом

Если вы не занимались организацией сетей или сетевой безопасностью, то работа Интернета может показаться вам немного таинственной. Мы не задумываемся о том, что, собственно, сеть делает каждый раз, когда мы открываем браузер и переходим на http://google.com, да и сейчас это нам не нужно. На самом деле я бы назвала фантастикой тот факт, что компьютерные интерфейсы достигли такого совершенства, что большинство пользователей Интернета не имеют ни малейшего представления о том, как он работает.

Однако скрапинг следует рассматривать не только как веб-интерфейс, лишь на уровне браузера (в плане обработки всех этих HTML, CSS и JavaScript), он также связан с типом сетевого соединения.

Чтобы дать вам некоторое представление об инфраструктуре, которая используется для загрузки информации в ваш браузер, приведем следующий пример. У Алисы есть веб-сервер. Боб использует настольный компьютер, который пытается подключиться к серверу Алисы. Когда одна машина хочет подсоединиться к другой, происходит следующий обмен:

1. Компьютер Боба посылает последовательность битов, представленных в виде низкого и высокого напряжений. Запрос Боба

разбит на фрагменты, к каждому фрагменту добавлен заголовок со служебной информацией (этим заведует протокол TCP). Передачей отдельных фрагментов от компьютера Боба до компьютера Алисы заведует протокол IP.

2. Локальный маршрутизатор Боба получает эту последовательность и интерпретирует ее как пакет с помощью собственного MAC-адреса и направляет на IP-адрес Алисы. Маршрутизатор заменяет в заголовке пакета обратный адрес на свой и посылает пакет дальше.
3. Пакет Боба проходит несколько промежуточных серверов, которые направляют его по правильному физическому/проводному пути на сервер Алисы.
4. Сервер Алисы получает пакет на свой IP-адрес.
5. Сервер Алисы считывает порт назначения пакета (почти всегда это порт 80 для веб-приложений, это что-то вроде «номера квартиры» в пакетной передаче данных, где IP-адрес является «улицей») в заголовке и передает его в соответствующее приложение – приложение веб-сервера.
6. Веб-сервер принимает поток данных от серверного процессора. Эти данные говорят что-то вроде:
   - это GET-запрос;
   - запрашивается следующий файл: index.html.
7. Веб-сервер находит соответствующий HTML-файл, записывает его в новый пакет для отправки Бобу и посылает его через свой локальный маршрутизатор обратно на компьютер Боба точно таким же вышеописанным способом.

И вуаля! У нас есть Интернет.

Итак, где в этом обмене задействован браузер? Абсолютно нигде. На самом деле браузеры – это относительно недавнее изобретение в истории Интернета, первый браузер Nexus появился в 1990 году.

Да, веб-браузер – очень полезная программа для создания пакетов информации, их отправки и интерпретации в виде красивых картинок, звуков, видео и текста. Однако веб-браузер – это просто код, а код можно разбить на части, выделить основные компоненты, перезаписать, повторно использовать и вообще сделать все, что угодно. Веб-браузер сообщает процессору о том, что нужно отправить некоторые данные в приложение, которое использует беспроводной (или проводной) интерфейс, однако библиотеки, предлагаемые различными языками программирования, могут выполнить то же самое.

Теперь покажем, как это делается в Python:

```
from urllib.request import urlopen
html = urlopen("http://pythonscraping.com/pages/page1.html")
print(html.read())
```

Вы можете сохранить этот код как *scrapetest.py* и запустить его в своем терминале с помощью команды:

```
$python scrapetest.py
```

Обратите внимание, если у вас дополнительно установлен Python 2.x, вам, возможно, потребуется явно вызывать Python 3.x с помощью команды:

```
$python3 scrapetest.py
```

Она выведет полный HTML-код страницы http://bit.ly/1QjYgcd. Если говорить более точно, она выводит HTML-файл *page1.html*, найденный в каталоге *<web root>/pages* на сервере, расположенном в домене http://pythonscraping.com.

В чем заключается разница? Большинство современных веб-страниц содержат файлы различных форматов. Это могут быть файлы изображений, файлы JavaScript, CSS-файлы или любой другой контент, который содержит запрашиваемая страница. Когда веб-браузер находит тег, например `<img src="cuteKitten.jpg">`, он отправляет еще один запрос на сервер, чтобы получить данные из файла *cuteKitten.jpg* для корректного отображения страницы. Имейте в виду, что наш скрипт Python пока не может вернуться обратно и запросить несколько файлов, он может прочитать только один запрошенный нами HTML-файл.

Итак, как это сделать? Строка

```
from urllib.request import urlopen
```

делает то, что написано, – импортирует функцию `urlopen` из модуля `request` библиотеки `urllib`.

urllib или urllib2?

Используя библиотеку urllib2 в Python 2.x, вы, возможно, заметили некоторые отличия между urllib2 и urllib. В Python 3.x urlib2 была переименована в urllib и разбита на несколько подмодулей: `urllib.request`, `urllib.parse` и `urllib.error`. Несмотря на то что названия функций преимущественно остались теми же, вам, возможно, стоит узнать, какие функции включены в подмодули новой urllib.

`urllib` – стандартная библиотека Python (то есть вам не нужно устанавливать ничего лишнего, чтобы запустить этот пример) и со-

держит функции для запроса данных в сети, обработки cookies и даже изменения метаданных (заголовков и пользовательского агента). На протяжении всей книги мы будем использовать `urllib` довольно часто, поэтому рекомендуем вам прочитать документацию Python по этой библиотеке (*http://bit.ly/1FncvYE*).

Функция `urlopen` используется, чтобы открыть удаленный объект в сети и прочитать его. Поскольку она имеет достаточно широкое применение (она может с легкостью прочитать HTML-файлы, файлы изображений или любой другой поток файлов), мы будем использовать ее довольно часто на протяжении всей книги.

Введение в BeautifulSoup

«Красивый суп, столь густой и зеленый,
Ожидающий в горячем глубоком блюде!
Кто для таких лакомств не наклонился бы?
Вечерний суп, красивый суп!»

Библиотека BeautifulSoup была названа в честь одноименного стихотворения, входящего в сказку Льюиса Кэролла «Приключения Алисы в Стране чудес». В сказке это стихотворение произносит персонаж по имени Черепаха Квази (Mock Turtle)[1].

Как и ее тезка, библиотека BeautifulSoup пытается придать смысл бессмыслице, она помогает отформатировать и систематизировать грязные интернет-данные, исправляя плохо размеченные HTML-страницы и выводя их в виде легко обрабатываемых объектов Python, представляющих собой XML-структуры.

Установка BeautifulSoup

Поскольку библиотека BeautifulSoup не является библиотекой Python по умолчанию, ее нужно установить. На протяжении всей книги мы будем использовать библиотеку BeautifulSoup 4 (также известную как BS4). Полные инструкции по установке BeautifulSoup 4 можно найти на Crummy.com. Однако для Linux основным методом будет:

```
$sudo apt-get install python-bs4
```

а для Mac:

```
$sudo easy_install pip
```

---

[1] Mock Turtle Soup – название блюда-подделки, популярного в Викторианскую эпоху, которое имитировало дорогой черепаший суп, но готовилось из телятины.

Эта команда устанавливает питоновский менеджер пакетов *pip*. Затем выполните следующее:

```
$pip install beautifulsoup4
```

чтобы установить библиотеку.

И снова обратите внимание, что если вы используете на вашем компьютере Python 2.x и 3.x, вам, возможно, потребуется явно вызвать python3:

```
$python3 myScript.py
```

Убедитесь, что используете `python3` при установке пакетов или пакеты можно установить под Python 2.x, а не под Python 3.x:

```
$sudo python3 setup.py install
```

При использовании pip вы можете также вызвать `pip3`, чтобы установить пакеты для Python 3.x:

```
$pip3 install beautifulsoup4
```

Установка пакетов в Windows практически идентична установке пакетов для Mac и Linux. Загрузите последнюю версию BeautifulSoup 4, воспользовавшись вышеприведенной ссылкой, зайдите в распакованную папку и выполните:

```
>python setup.py install
```

Вот и все! BeautifulSoup теперь используется в качестве библиотеки Python на вашем компьютере. Вы можете проверить это, открыв терминал Python и импортировав библиотеку:

```
$python
> from bs4 import BeautifulSoup
```

Импорт должен быть выполнен без ошибок.

Кроме того, есть exe-инсталлятор для *pip* под Windows, поэтому вы можете легко установить пакеты и работать с ними:

```
>pip install beautifulsoup4
```

Сохранение библиотек с помощью виртуальных окружений

Если вам необходимо работать одновременно с несколькими проектами Python, или вам нужен способ, с помощью которого можно легко связать проекты с использованными библиотеками, или вы хотите избежать возможных конфликтов между уже имеющимися библиотеками, можно установить виртуальную среду Python, чтобы хранить проекты по отдельности и с легкостью управлять ими.

Если вы устанавливаете библиотеку Python без виртуального окружения, вы устанавливаете ее *глобально*. Обычно для этого надо иметь права администратора или root-пользователя. К счастью, создать виртуальное окружение довольно просто:

```
$ virtualenv scrapingEnv
```

Эта команда создает новое окружение *scrapingEnv*, которое нужно активировать для использования:

```
$ cd scrapingEnv/
$ source bin/activate
```

После активации окружения вы увидите, что название окружения появилось в командной строке, напоминая, что теперь вы работаете с ним. Любые установленные библиотеки или запущенные скрипты теперь будут храниться и выполняться только в данном виртуальном окружении.
Работая в только что созданном окружении scrapingEnv, я могу установить и использовать библиотеку BeautifulSoup, например:

```
(scrapingEnv)ryan$ pip install beautifulsoup4
(scrapingEnv)ryan$ python
> from bs4 import BeautifulSoup
>
```

Я могу покинуть окружение с помощью команды деактивации, после чего у меня уже не будет доступа к библиотекам, установленным внутри этого окружения:

```
(scrapingEnv)ryan$ deactivate
ryan$ python
> from bs4 import BeautifulSoup
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ImportError: No module named 'bs4'
```

Изолирование библиотек для каждого отдельного проекта позволяет заархивировать все окружение в папку и отправить ее кому-то еще. Если у этих пользователей на их компьютерах установлена та же самая версия Python, они смогут работать с вашим кодом в данной виртуальной среде, не прибегая к инсталляции других библиотек.
Несмотря на то что в рамках этой книги мы не будем явно рассказывать, как использовать виртуальное окружение, имейте в виду, что вы можете воспользоваться виртуальным окружением в любой момент, заранее активировав его.

Запуск BeautifulSoup

Наиболее часто используемым объектом в библиотеке BeautifulSoup является, соответственно, объект BeautifulSoup. Давайте посмотрим на него в действии, изменив пример, приведенный в начале этой главы:

```
from urllib.request import urlopen
from bs4 import BeautifulSoup
html = urlopen("http://www.pythonscraping.com/pages/page1.html")
bsObj = BeautifulSoup(html.read());
print(bsObj.h1)
```

Вывод выглядит так:

```
<h1>An Interesting Title</h1>
```

Как и в примере выше, мы импортируем библиотеку `urlopen` и вызываем функцию `html.read()`, чтобы получить HTML-контент страницы. Затем этот HTML-контент мы преобразуем в объект BeautifulSoup со следующей структурой:

- **html** → <html><head>...</head><body>...</body></html>
  - **head** → <head><title>A Useful Page<title></head>
    - **title** → <title>A Useful Page</title>
  - **body** → <body><h1>An Int...</h1> Lorem ip... </body>
    - **h1** → <h1>An Interesting Title</h1>
    - **div** → Lorem Ipsum dolor...

Обратите внимание, что тег `<h1>`, который мы извлекли из страницы, был вложен в два слоя внутри объекта BeautifulSoup (html → body → h1). Однако после излечения тега h1 из объекта мы обращаемся к нему напрямую:

```
bsObj.h1
```

Фактически любой вызов функции из нижеприведенного списка сгенерирует один и тот же вывод:

```
bsObj.html.body.h1
bsObj.body.h1
bsObj.html.h1
```

Мы надеемся, что этот небольшой пример использования `BeautifulSoup` дал вам представление о возможностях и простоте использования этой библиотеки. Практически любую информацию можно извлечь из любого файла HTML (или XML), при условии что она заключена в определенный идентифицирующий тег или расположена рядом с ним. В главе 3 мы подробнее разберем вызов более сложных функций BeautifulSoup, а также рассмотрим регулярные выражения и покажем, как их можно использовать вместе с `BeautifulSoup` для извлечения информации с веб-сайтов.

Как обеспечить надежный скрапинг

Веб не идеален. Данные плохо отформатированы, веб-сайты падают, закрывающие теги отсутствуют. Самая неприятная ситуация, которая случается при использовании веб-скрапера, – вы пошли спать, запустив скрапер в надежде, что на следующий день он соберет все данные в базу, однако, проснувшись, видите, скрапер выдал ошибку при обработке данных неизвестного формата и прекратил свою работу вскоре после того, как вы оторвали свой взгляд от монитора. В таких ситуациях вы, возможно, проклинаете разработчика веб-сайта (и странно отформатированные данные), но на самом деле обвинять вы должны самого себя, изначально не надеясь на счастливый случай!

Давайте взглянем на первую строку нашего скрапера (расположена после операторов импорта) и выясним, как обработать сгенерированные исключения:

```
html = urlopen("http://www.pythonscraping.com/pages/page1.html")
```

Есть две основные ошибки, которые могут возникнуть здесь:

- страница не найдена на сервере (или есть какая-то ошибка при ее получении);
- сервер не найден.

В первом случае будет возвращена ошибка HTTP. Ошибкой HTTP может быть «404 Page Not Found», «500 Internal Server Error» и т. д. Во всех этих случаях функция `urlopen` генерирует общее исключение «HTTPError». Мы можем обработать это исключение следующим образом:

```
try:
    html = urlopen("http://www.pythonscraping.com/pages/page1.html")
except HTTPError as e:
    print(e)
    #возвратить null, прервать или выполнять операции по "Плану Б"
else:
    #программа продолжает работу. Примечание: если возвращаете или
    прерываете в #exception catch, оператор "else" использовать
    не нужно
```

Если вернулся код ошибки HTTP, программа выводит сообщение об ошибке, и оставшаяся часть программы для оператора `else` не выполняется.

Если сервер не найден вообще (например, http://www.pythonscraping.com упал или URL-адрес набран неправильно), функция `urlopen` возвращает объект `None`. Этот объект аналогичен значению `null`, исполь-

зуемому в других языках программирования. Дополнительно мы можем проверить, является ли возвращенная html-страница объектом None:

```
if html is None:
    print("URL is not found")
else:
    #программа продолжает работу
```

Конечно, даже если страница успешно получена с сервера, остается еще вопрос, связанный с тем, что контент страницы может совершенно не соответствовать нашим ожиданиям. Каждый раз, когда вы обращаетесь к тегу в объекте BeautifulSoup, полезно удостовериться в том, что этот тег существует на самом деле. Если вы попытаетесь обратиться к тегу, который не существует, BeautifulSoup вернет объект None. При попытке получить тег объекта None будет сгенерировано исключение AttributeError.

Следующая строка (где nonExistentTag – это выдуманный тег, а не название действительной функции BeautifulSoup):

```
print(bsObj.nonExistentTag)
```

возвращает объект None. Этот объект вполне пригоден для обработки и проверки. Проблема возникнет, если вы, вместо того чтобы проверить объект, продолжите работу и попытаетесь вызвать некоторые другие функции для объекта None, как показано в следующей строке:

```
print(bsObj.nonExistentTag.someTag)
```

которая вернет исключение:

```
AttributeError: 'NoneType' object has no attribute 'someTag'
```

Итак, как мы можем избежать этих двух ситуаций? Самый простой способ – явно протестировать возможность возникновения обеих ситуаций:

```
try:
    badContent = bsObj.nonExistingTag.anotherTag
except AttributeError as e:
    print("Tag was not found")
else:
    if badContent == None:
       print ("Tag was not found")
    else:
        print(badContent)
```

Проверка и обработка каждой ошибки кажется поначалу трудоемкой, однако этот код легко реорганизовать, сделать простым в написании (и, что более важно, менее трудным для чтения). Например, код, приведенный ниже, – это тот же самый наш скрапер, записанный немного по-другому:

```
from urllib.request import urlopen
from urllib.error import HTTPError
from bs4 import BeautifulSoup
 def getTitle(url):
    try:
        html = urlopen(url)
    except HTTPError as e:
        return None
    try:
        bsObj = BeautifulSoup(html.read())
        title = bsObj.body.h1
    except AttributeError as e:
        return None
    return title
title = getTitle("http://www.pythonscraping.com/pages/page1.html")
if title == None:
    print("Title could not be found")
else:
    print(title)
```

В этом примере мы создали функцию `getTitle`, которая возвращает либо название страницы, либо объект `None`, если с извлечением страницы возникла какая-то проблема. Как и в предыдущем примере, внутри `getTitle` мы проверяем возможность получения `HTTPError`, а также инкапсулируем две строки BeautifulSoup в оператор `try`. `AttributeError` может быть сгенерирована любой из этих строк (если сервер не существует, `html` будет объектом `None`, а `html.read()` сгенерирует `AttributeError`). На самом деле мы могли бы включить в оператор `try` столько строк, сколько нам нужно, или вызвать вообще другую функцию, которая может сгенерировать `AttributeError` в любой момент.

При написании скрапера важно позаботиться об общей структуре Вашего кода, чтобы обработать исключения и сделать его читаемым. Кроме того, вы, вероятно, захотите повторно использовать код. С помощью функций `getSiteHTML` и `getTitle` (в сочетании с тщательной обработкой исключений) скрапинг веб-сайтов становится быстрым и надежным.

Глава 2

Продвинутый парсинг HTML

Когда Микеланджело спросили, как он смог так мастерски вылепить Давида, он ответил: «Это легко. Вы просто отсекаете камень, который не похож на Давида».

Хотя веб-скрапинг во многих аспектах отличается от ваяния в мраморе, мы должны принять схожую установку, когда дело доходит до извлечения информации со сложных веб-страниц. Есть много методов, которые отсекают ненужный контент до тех пор, пока мы не найдем интересующую нас информацию. В этой главе мы расскажем о парсинге сложных HTML-страниц, предназначенном для извлечения только интересующей нас информации.

Вам не всегда нужен молоток

Когда сталкиваешься с гордиевым узлом тегов, всегда есть соблазн сразу начать использовать многострочные операторы в попытке извлечь нужную информацию. Однако имейте в виду, что сумбурное применение методов, описанных в этом разделе, с безудержной энергией может привести к написанию кода, который потом трудно будет отладить и использовать. Перед началом работы давайте взглянем на некоторые способы, которые позволят вам избежать необходимости прибегать к продвинутому HTML-парсингу!

Скажем, вас интересует определенный контент. Это может быть имя, статистические данные или блок текста. Возможно, оно вложено в тег 20-го уровня внутри HTML-каши, в которой невозможно найти какие-либо полезные теги или атрибуты HTML. Давайте сразу напишем следующую строку, чтобы попытаться извлечь информацию:

```
bsObj.findAll("table")[4].findAll("tr")[2].find("td").findAll("div")[1].find("a")
```

Выглядит не очень. Помимо внешнего вида строки, даже малейшее изменение сайта администратором может полностью нарушить работу Вашего веб-скрапера. Какие у вас есть варианты?

- Взгляните на ссылку «print this page» или мобильную версию сайта, у которой более понятная HTML-структура (более подробно о работе с мобильных устройств и получении мобильных версий сайтов читайте в главе 12).
- Изучите информацию, скрытую в файле JavaScript. Для этого вам, возможно, потребуется исследовать импортируемые файлы JavaScript. Например, однажды я собрала адреса улиц (вместе с широтой и долготой) с веб-сайта в аккуратно отформатированный массив, исследуя JavaScript-скрипты Google Maps, которые выводят поверх каждого адреса маркеры в виде приколотых булавок.
- Информацию можно получить не только из названий страниц, но и из URL-адреса самой страницы.
- Если по какой-то причине интересующую вас информацию можно найти только на данном сайте, вам не повезло. Если нет, подумайте о других источниках получения информации. Есть еще один веб-сайт с такими же данными? Этот сайт содержит данные, которые собраны или агрегированы с другого сайта?

Когда вы сталкиваетесь с глубоко закопанными или плохо отформатированными данными, не беритесь сразу же делать скрапинг. Сделайте глубокий вдох и подумайте об альтернативных вариантах. Если вы уверены, что альтернатив нет, то оставшаяся часть этой главы предназначена для вас.

Еще одно применение BeautifulSoup

В главе 1 мы кратко рассказали об установке и запуске BeautifulSoup, а также об однократном отборе одного объекта. В этом разделе мы расскажем о поиске тегов по атрибутам, работе со списками тегов, а также навигации по дереву синтаксического разбора.

Почти каждый веб-сайт, который вам предстоит обработать, содержит таблицы стилей. Несмотря на то что вы, возможно, думаете, что стилевое оформление, специально предназначенное для браузера и облегчения интерпретации контента, является плохой идеей, CSS на самом деле упрощает работу веб-скрапера. CSS основан на дифференциации HTML-элементов, которые могут иметь точно такую

же разметку, но отличаться по стилю. То есть некоторые теги могут выглядеть следующим образом:

```
<span class="green"></span>
```

тогда как другие теги выглядят так:

```
<span class="red"></span>
```

Веб-скраперы могут легко дифференцировать два этих различных тега в зависимости от их класса. Например, они могут использовать BeautifulSoup, чтобы извлечь весь красный текст, не трогая при этом зеленый. Поскольку CSS опирается на эти идентифицирующие атрибуты, задающие стилевое оформление сайтов, вы можете почти быть уверены, что эти классы и ID-атрибуты широко используются большинством современных веб-сайтов.

Давайте создадим веб-скрапер, который просканирует страницу, расположенную на http://bit.ly/1Ge96Rw.

На этой странице прямая речь персонажей выделена красным цветом, тогда как имена персонажей выделены зеленым цветом. В примере, приведенном ниже, где приводится исходный код страницы, вы можете увидеть теги span, которые ссылаются на соответствующие классы CSS:

```
"<span class="red">Heavens! what a virulent attack!</span>" replied <span class=
"green">the prince</span>, not in the least disconcerted by this reception.
```

Мы можем захватить всю страницу и создать объект BeautifulSoup с ней, используя программу, аналогичную той, что использовали в главе 1:

```
from urllib.request import urlopen
from bs4 import BeautifulSoup
html = urlopen("http://www.pythonscraping.com/pages/warandpeace.html")
bsObj = BeautifulSoup(html)
```

С помощью этого объекта BeautifulSoup мы можем вызвать функцию findAll, чтобы извлечь питоновский список имен собственных, их находят путем отбора текста, находящегося внутри тегов (findAll является чрезвычайно гибкой функцией, которую мы будем еще много использовать в дальнейшем в этой книге):

```
nameList = bsObj.findAll("span", {"class":"green"})
for name in nameList:
    print(name.get_text())
```

При запуске она должна перечислить все имена собственные в порядке их появления в *War and Peace*. Итак, что здесь происходит? Ра-

нее мы вызвали `bsObj.tagName`, чтобы получить первое вхождение этого тега на данной странице. Теперь мы вызываем `bsObj.findAll(tagName, tagAttributes)`, чтобы получить список всех тегов на странице, а не только первый.

Получив список имен, программа перебирает все имена в списке и печатает `name.get_text()`, чтобы отделить контент от тегов.

Когда использовать get_text() и когда сохранять теги

`.get_text()` удаляет все теги из документа, с которым вы работаете, и возвращает строку, содержащую только текст. Например, если вы работаете с большим блоком текста, который содержит много гиперссылок, параграфов и тегов, все эти элементы будут удалены, и вы получите блок текста без тегов.

Запомните, что гораздо проще найти интересующую информацию в объекте BeautifulSoup, чем в блоке текста. Вызов функции `.get_text()` нужно всегда делать в последнюю очередь, непосредственно перед печатью, сохранением и управлением итоговыми данными. В общем, вы должны постараться максимально дольше сохранить структуру тегов документа.

find() и findAll()

`find()` и `findAll()` – это две функции BeautifulSoup, которые вы, вероятно, будете использовать чаще всего. С их помощью вы можете легко отфильтровать HTML-страницы, чтобы найти списки нужных тегов или отдельный тег в зависимости от их атрибутов.

Эти две функции очень схожи между собой, о чем свидетельствует их описание в документации BeautifulSoup:

```
findAll(tag, attributes, recursive, text, limit, keywords)
find(tag, attributes, recursive, text, keywords)
```

По всей вероятности, 95% времени вы будете тратить, используя первые два аргумента: `tag` и `attributes`. Однако давайте взглянем на все аргументы более подробно.

Аргумент `tag` – это тег, который мы уже видели раньше. Вы можете передать строковое имя тега или даже питоновский список имен тегов. Например, следующая строка вернет список всех тегов заголовков в документе[1]:

```
.findAll({"h1","h2","h3","h4","h5","h6"})
```

[1] Если вы хотите получить список всех тегов в документе, есть более экономичные способы написать код для выполнения данной операции. Мы рассмотрим другие способы решения подобных проблем в разделе «BeautifulSoup и регулярные выражения».

Аргумент attributes берет питоновский словарь атрибутов и ищет совпадения с тегами, которые содержат любой из этих атрибутов. Например, функция, приведенная ниже, возвращает теги span, задающие как красный, так и зеленый цвет шрифта в документе HTML:

```
.findAll("span", {"class":"green", "class":"red"})
```

Аргумент recursive является булевым значением. Как глубоко вы хотите просканировать документ? Если для аргумента recursive установлено значение True, функция findAll исследует элементы, которые являются потомками, потомками потомков тегов, соответствующих вашим параметрам. Если установлено значение False, то будут исследованы только теги верхнего (первого) уровня. По умолчанию findAll работает рекурсивно (для аргумента установлено значение True). Вообще, оптимальным будет оставить значение по умолчанию, если у вас действительно нет конкретного плана действий и эффективность работы под вопросом.

Аргумент text необычен тем, что он ищет совпадения, руководствуясь текстовым содержимым тегов, а не свойствами самих тегов. Например, если мы хотим найти частоту размещения слова «the prince» внутри тегов интересующей страницы, мы можем заменить нашу функцию .findAll(), взятую в предыдущем примере, следующими строками:

```
nameList = bsObj.findAll(text="the prince")
print(len(nameList))
```

Вывод равен «7».

Разумеется, аргумент limit используется только в методе findAll. Метод find эквивалентен вызову метода findAll с лимитом 1. Вы можете установить данное значение, если хотите извлечь лишь первые x элементов со страницы. Однако помните, что он выведет первые элементы в том порядке, в котором они встречаются в документе, и необязательно это будут элементы, которые вы искали.

Аргумент keyword позволяет вам выбрать теги, содержащие конкретный атрибут. Например:

```
allText = bsObj.findAll(id="text")
print(allText[0].get_text())
```

Предостережение по поводу аргумента keyword

Аргумент keyword может пригодиться в некоторых ситуациях. Однако в качестве характеристики BeautifulSoup он с технической точки зрения избыточен. Имейте в виду, что все, что можно выполнить с помощью

аргумента keyword, можно также выполнить с использованием методов, которые мы обсудим позже в этой главе (см. «Регулярные выражения» и «Лямбда-выражения»).
Например, две следующие строки идентичны:

```
bsObj.findAll(id="text")
bsObj.findAll("", {"id":"text"})
```

Кроме того, вы можете столкнуться с проблемами, используя аргумент keyword, главным образом при поиске элементов по атрибуту class, поскольку class является защищенным ключевым словом в Python. То есть class – это зарезервированное слово в Python, которое нельзя использовать в качестве имени переменной или аргумента (это не относится к аргументу keyword метода BeautifulSoup.findAll(), рассмотренному ранее)[1]. Например, если вы попытаетесь вызвать метод, приведенный ниже, то получите синтаксическую ошибку из-за нестандартного использования class:

```
bsObj.findAll(class="green")
```

Вместо этого вы можете использовать несколько неуклюжее решение BeautifulSoup, которое включает в себя добавление символа подчеркивания:

```
bsObj.findAll(class_="green")
```

Как вариант вы можете заключить class в кавычки:

```
bsObj.findAll("", {"class":"green"}
```

В этот момент вы можете спросить себя: «Но подождите, я ведь уже знаю, как получить список тегов с помощью атрибутов – передать атрибуты функции в список словарей?»

Напомним, что передача списка тегов в .findAll() с помощью списка атрибутов выполняет роль фильтра «or» (то есть отбирает список всех тегов, которые содержат tag1 или tag2 или tag3 ...). Если вы работаете с длинным списком тегов, вы можете столкнуться с большим количеством ненужной информации. Аргумент keyword позволяет добавить дополнительный фильтр «and», чтобы отфильтровать ее.

Другие объекты BeautifulSoup

До сих пор в этой книге вы сталкивались с двумя типа объектов библиотеки BeautifulSoup:

- объекты BeautifulSoup: смотрите предыдущие примеры кода, например bsObj;

[1] Пособие по языку Python содержит полный перечень защищенных ключевых слов.

- объекты Tag: получают списками или индивидуально путем вызова find и findAll для объекта BeautifulSoup:

```
bsObj.div.h1
```

Однако в библиотеке BeautifulSoup есть еще два объекта, которые хоть и используются реже, но все же важны:

- объекты NavigableString: используются для представления текста внутри тегов, а не самих тегов (некоторые функции вместо тегов работают с объектами типа NavigatableString, т. е. принимают в качестве аргумента и возвращают NavigatableString);
- Объект Comment: используются для поиска HTML-комментариев в тегах комментариев <!–как, например, этот-->.

Эти четыре объекта – единственные объекты, с которыми вы будете работать (на момент написания книги), используя библиотеку BeautifulSoup.

Навигация по дереву синтаксического разбора

Функция findAll осуществляет поиск тегов по их имени и атрибуту. Но что, если вам нужно найти тег, основываясь на его месторасположении в документе? Вот именно здесь и пригодится древовидная навигация. В главе 1 мы рассказывали о навигации в одну сторону по дереву синтаксического разбора BeautifulSoup:

```
bsObj.tag.subTag.anotherSubTag
```

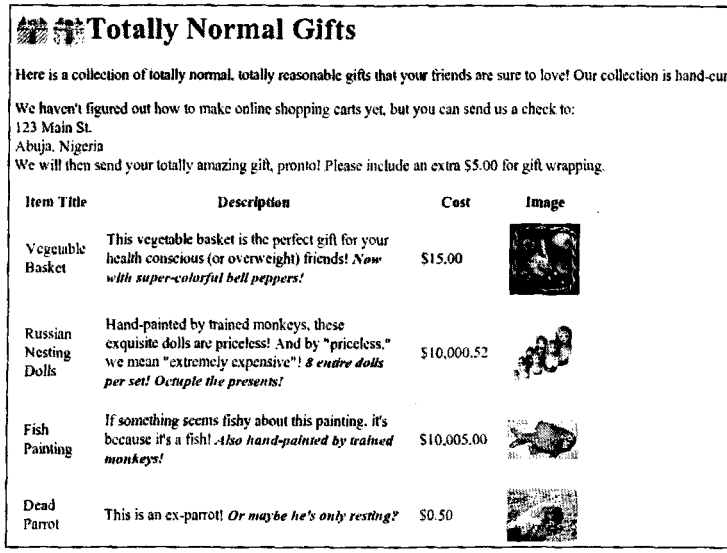
Теперь давайте расскажем, как осуществляется навигация вверх, поперек и диагонально по древовидной структуре HTML, используя в качестве страницы для скрапинга сайт интернет-магазина http://bit.ly/1KGe2Qk (см. рис. 2.1):

HTML этой страницы, представленный в виде дерева (некоторые теги опущены для краткости), выглядит следующим образом:

- html
  - body
    - div.wrapper
      - h1
      - div.content
      - table#giftList
        - tr
          - th
          - th
          - th

- th
- tr.gift#gift1
  - td
  - td
    - span.excitingNote
  - td
  - td
    - img
- ...продолжение строк таблицы...
- div.footer

Мы будем использовать эту же HTML-структуру в качестве примера в следующих нескольких разделах.



Totally Normal Gifts

Here is a collection of totally normal, totally reasonable gifts that your friends are sure to love! Our collection is hand-curate

We haven't figured out how to make online shopping carts yet, but you can send us a check to:
123 Main St.
Abuja, Nigeria
We will then send your totally amazing gift, pronto! Please include an extra $5.00 for gift wrapping.

Item Title	Description	Cost	Image
Vegetable Basket	This vegetable basket is the perfect gift for your health conscious (or overweight) friends! *Now with super-colorful bell peppers!*	$15.00	
Russian Nesting Dolls	Hand-painted by trained monkeys, these exquisite dolls are priceless! And by "priceless," we mean "extremely expensive"! *8 entire dolls per set! Octuple the presents!*	$10,000.52	
Fish Painting	If something seems fishy about this painting, it's because it's a fish! *Also hand-painted by trained monkeys!*	$10,005.00	
Dead Parrot	This is an ex-parrot! *Or maybe he's only resting?*	$0.50	

Рис. 2.1 ❖ Скриншот сайта http://www.pythonscraping.com/pages/page3.html

Работа с дочерними элементами и элементами-потомками

В библиотеке BeautifulSoup, а также во многих других библиотеках существует различие между *дочерними элементами (children)* и *элементами-потомками (descendants)*: так же, как в и генеалогическом дереве человека, дочерние теги всегда на один уровень ниже роди-

тельского тега, в то время как теги-потомки могут быть ниже родительского тега, располагаясь на любом уровне. Например, теги tr являются дочерними по отношению к тегу table, в то время как теги tr, th, td, img и span являются по отношению к тегу table потомками (по крайней мере, для нашей страницы, приведенной в качестве примера). Все дети являются потомками, но не все потомки – дети.

В целом функции BeautifulSoup всегда работают с потомками выбранного тега. Например, bsObj.body.h1 выбирает первый тег h1, который является потомком тега body. Она не найдет теги, расположенные вне body.

Аналогично bsObj.div.findAll("img") найдет первый тег div в документе, а затем извлечет все теги img, которые являются потомками тега div.

Если вы хотите извлечь только дочерние теги, можно использовать тег .children:

```
from urllib.request import urlopen
from bs4 import BeautifulSoup

html = urlopen("http://www.pythonscraping.com/pages/page3.html")
bsObj = BeautifulSoup(html)

for child in bsObj.find("table",{"id":"giftList"}).children:
    print(child)
```

Этот код выводит список строк с названиями продуктов таблицы giftList. Если бы вы написали его, используя функцию descendants() вместо функции children(), в таблице было бы найдено и напечатано около двух десятков тегов, в том числе теги img, теги span и отдельные теги td. Безусловно, важно различать детей и потомков!

Работа с одноуровневыми элементами

Функция next_siblings() библиотеки BeautifulSoup упрощает сбор данных из таблиц, особенно из таблиц со строками-заголовками:

```
from urllib.request import urlopen
from bs4 import BeautifulSoup
html = urlopen("http://www.pythonscraping.com/pages/page3.html")
bsObj = BeautifulSoup(html)

for sibling in bsObj.find("table",{"id":"giftList"}).tr.next_siblings:
    print(sibling)
```

Этот код печатает все строки с названиями продуктов из таблицы, за исключением первой строки заголовка. Почему название строки пропущено? Две причины: во-первых, объекты не могут быть одно-

уровневыми элементами по отношению друг к другу. Каждый раз, когда вы получаете одноуровневые элементы объекта, сам объект в список не включается. Во-вторых, эта функция вызывает только *следующие* одноуровневые элементы. Если бы мы отобрали строку в середине списка и вызвали для нее `next_siblings`, были бы возвращены только следующие одноуровневые элементы. Таким образом, выбрав строку заголовка и вызвав `next_siblings`, мы можем выбрать все строки в таблице, не выбирая при этом строку заголовка.

Конкретизируйте выбор

Обратите внимание, что вышеприведенный код будет работать так же хорошо, если мы возьмем `bsObj.table.tr` или даже просто `bsObj.tr` для отбора первой строки таблицы. Тем не менее довольно затруднительно записать все в более развернутом виде:

```
bsObj.find("table",{"id":"giftList"}).tr
```

Даже если мы работаем с одной таблицей (или другим целевым тегом) на странице, легко что-то пропустить. Кроме того, макеты страниц все время меняются. Тег, который располагался первым на странице, завтра может стать вторым или третьим на странице. Чтобы повысить надежность работы скрапера, нужно максимально конкретизировать выбор тегов. По возможности используйте атрибуты тега.

В качестве дополнения к функции `next_siblings` функция `previous_siblings` применяется в тех случаях, когда в конце списка извлекаемых одноуровневых тегов располагается легко выделяемый тег.

И конечно же, есть функции `next_sibling` и `previous_sibling`, которые выполняют практически ту же функцию, что и `next_siblings` и `previous_siblings`, за исключением того, что они возвращают один тег, а не список тегов.

Работа с родительскими элементами

Вероятно, в ходе скрапинга страниц родительские элементы вас будут интересовать реже, чем дочерние или одноуровневые элементы.

Как правило, когда мы исследуем HTML-страницы с целью их краулинга, сначала смотрим теги верхнего уровня, а затем выясняем, как проложить наш путь в глубину, чтобы максимально точно извлечь интересующие куски информации. Однако иногда вы можете оказаться в нестандартной ситуации, которая потребует от вас использования функций `.parent` и `.parents`., осуществляющих поиск родительских элементов. Например:

```
from urllib.request import urlopen
from bs4 import BeautifulSoup
```

```
html = urlopen("http://www.pythonscraping.com/pages/page3.html")
bsObj = BeautifulSoup(html)
print(bsObj.find("img",{"src":"../img/gifts/img1.jpg"
                        }).parent.previous_sibling.get_text())
```

Этот код печатает цену объекта, представленного в виде изображения, расположенного по адресу `../img/gifts/img1.jpg` (в данном случае цена равна «$ 15.00»).

Как он работает? Диаграмма, приведенная ниже, представляет собой пронумерованную древовидную структуру фрагмента HTML-страницы, с которым мы в данный момент работаем:

- <tr>
  - <td>
  - <td>
  - <td> **(3)**
    - "$15.00" **(4)**
  - s<td> **(2)**
    - <img src="../img/gifts/img1.jpg"> **(1)**

1. Сначала выбираем тег изображения, содержащий атрибут `src="../img/gifts/img1.jpg"`.
2. Выбираем родительский тег для него (в данном случае тег `<td>`).
3. Выбираем `previous_sibling` для тега `<td>` (в данном случае тег `<td>`, который содержит стоимость продукта в долларах).
4. Выбираем *текст внутри этого тега*, «$ 15.00».

Регулярные выражения

Есть старая компьютерная шутка: «Допустим, у вас есть проблема и вы хотите ее решить с помощью регулярных выражений. Итак, теперь у вас две проблемы».

К сожалению, регулярные выражения (часто используется сокращение *regex*) составляют с использованием больших таблиц символов, которые, будучи соединенными вместе, выглядят как бессмыслица. Это обычно отпугивает людей, а потом они тратят свое рабочее время на написание излишне сложных функций поиска и фильтрации, когда изначально можно было обойтись одной строкой регулярного выражения!

К счастью для вас, не все регулярные выражения сложны в написании, их можно быстро запустить и легко изучить, поэкспериментировав с несколькими простыми примерами.

Регулярные выражения называются так потому, что они используются для поиска периодических строк (последовательностей). Они могут вынести окончательный вердикт «Да, эта строка, которую вы мне дали,

соответствует правилам, и я возвращаю ее» или «Эта строка не соответствует правилам, и я отклоняю ее». Это исключительно удобно для быстрого сканирования больших документов, чтобы найти строки, представленные в виде телефонных номеров или адресов электронной почты.

Обратите внимание, что я использовала термин *периодическая строка (regular string)*. Что представляет собой периодическая строка? Это любая строка, которую можно получить с помощью ряда линейных правил типа[1]:

1. Запишите букву «a», по крайней мере, один раз.
2. Добавьте к этой букве букву «b» ровно пять раз.
3. Добавьте к этой букве букву «c» любое четное количество раз.
4. Опционально запишите букву «d» в конце.

Строки, которые соответствуют этим правилам: «aaaabbbbbccccd», «aabbbbbcc» и т. д. (существует бесконечное число вариаций).

Регулярные выражения – это всего лишь быстрый способ представления этих наборов правил. Например, ниже приводится регулярное выражение для ряда вышеописанных шагов:

```
aa*bbbbb(cc)*(d | )
```

Эта строка может показаться немного сложной на первый взгляд, но она станет понятнее, когда мы разобьем ее на части:

*aa**

Записывает букву *a*, после нее следует ***a**** (читается как *a звездочка*), что означает «любое количество букв *a*, в том числе и ноль». Таким образом, мы можем гарантировать, что буква *a* будет записана, по крайней мере, один раз.

bbbbb

Здесь нет никаких специальных оговорок – просто пять букв *b*, расположенных в ряд.

*(cc)**

Любое четное количество элементов можно сгруппировать в пары. Поэтому, чтобы выполнить это правило применительно к четному

---

[1] Вы можете спросить: «А существуют ли нерегулярные выражения?» Нерегулярные выражения выходят за рамки этой книги, они содержат строки типа «выберите простое число x, запишите x повторений буквы ***a***, потом 2*x повторений буквы ***b***» или «запишите палиндром». С помощью регулярного выражения найти такие строки невозможно. К счастью, у меня никогда не было ситуации, когда моему веб-скраперу требовалось найти строки подобного рода.

количеству элементов, вы можете записать две буквы *c*, заключить их в скобки и поставить после них символ звездочки. Это означает, что у вас может быть любое количество *пар* букв *c* (включая также нулевое количество пар).

(d|)

Вертикальная черта в середине двух выражений означает возможность выбора («этот символ или тот»). В этом случае мы задаем правило «добавить букву *d* с последующим пробелом *или* просто добавить пробел без буквы *d*». Таким образом, мы можем гарантировать, что у нас будет максимум одна буква *d* с последующим пробелом, завершающим строку.

Экспериментируйте с ReGex

Изучая регулярные выражения, важно поэкспериментировать с ними и получить представление, как они работают.
Если вы плохо представляете, как запустить редактор кода, написать несколько строк и запустить программу, чтобы убедиться в том, что регулярное выражение работает правильно, можно перейти на веб-сайт, например http://www.regexpal.com/, и протестировать свои регулярные выражения.

Один из классических примеров применения регулярных выражений – поиск адресов электронной почты. Несмотря на то что точные правила, задающие адреса электронной почты, незначительно отличаются для различных почтовых серверов, мы можем создать несколько общих правил. Соответствующее регулярное выражение для каждого из этих правил приведено во втором столбце:

Правило 1 Первая часть электронного адреса содержит, по крайней мере, что-то одно из нижеперечисленного: заглавные буквы, строчные буквы, числа 0–9, точки (.), знаки плюса (+) или символы подчеркивания (_)	**[A-Za-z0-9\._+]+** Регулярное выражение, записанное стенографией, выглядит довольно заумно. Например, известно, что «A-Z» означает «любая заглавная буква от A до Z». Заключив все возможные последовательности и символы в квадратные скобки (в противоположность круглым скобкам), мы говорим «этот символ может быть одним из этих элементов, указанных в квадратных скобках». Также обратите внимание на то, что знак + означает «эти символы могут встречаться любое количество раз, но не менее одного раза»
Правило 2 Затем в электронном адресе ставится символ @	**@** Тут все просто: символ @ должен располагаться посередине и используется ровно один раз

Правило 3 Теперь электронный адрес должен содержать, по крайней мере, одну заглавную или строчную букву	**[A-Za-z]+** После символа @ в первой части доменного имени мы можем использовать лишь буквы. Кроме того, нужно задать хотя бы один символ
Правило 4 Затем ставится точка (.)	**\.** Вы должны поставить точку (.) перед доменным именем
Правило 5 В итоге электронный адрес заканчивается на com, org, edu или net (на практике используются различные домены верхнего уровня, но этих четырех будет достаточно для примера)	**(com|org|edu|net)** Нужно перечислить возможные последовательности букв, которые могут встретиться после точки во второй части электронного адреса

Выполнив конкатенацию всех правил, мы получим регулярное выражение:

```
[A-Za-z0-9\._+]+@[A-Za-z]+\.(com|org|edu|net)
```

При попытке написать любое регулярное выражение с нуля лучше сначала составить список шагов, в которых конкретно будет указано, как должна выглядеть нужная строка. Обратите внимание на особые случаи. Например, при поиске телефонных номеров вы будете учитывать международные коды и добавочные номера?

В табл. 2.1 перечислены некоторые наиболее часто используемые символы регулярных выражений с кратким пояснением и примером. Этот список далеко не полон, и, как упоминалось ранее, вы можете столкнуться с небольшими изменениями от языка к языку. Однако эти 12 символов наиболее часто используются в регулярных выражениях Python. Их можно использовать для поиска и сбора любых типов строк.

Таблица 2.1. Часто используемые символы регулярных выражений

Символ(ы)	Значение	Пример	Найденные совпадения
*	Ищет 0 или больше совпадений с предыдущим символом, подвыражением или символом в квадратных скобках	a*b*	aaaaaaaa, aaabbbbb, bbbbbb
+	Ищет 1 или больше совпадений с предыдущим символом, подвыражением или символом в квадратных скобках	a+b+	aaaaaaaab, aaabbbbb, abbbbbb
[]	Ищет совпадения с любым символом, указанным в квадратных скобках (т. е. «возьмите любой из этих символов»)	[A-Z]*	APPLE, CAPITALS, QWERTY

Окончание табл. 2.1

Символ(ы)	Значение	Пример	Найденные совпадения
()	Сгруппированное подвыражение (оценивается первым в соответствии с «порядком выполнения операций» регулярных выражений)	(a*b)*	aaabaab, abaaab, ababaaaaab
{m, n}	Ищет от m до n совпадений (включительно) предыдущего символа, подвыражения или символа в квадратных скобках	a{2,3}b{2,3}	aabbb, aaabbb, aabb
[^]	Ищет совпадения с любым отдельным символом, заданным *вне* квадратных скобок	[^A-Z]*	apple, lowercase, qwerty
|	Ищет совпадения с любым символом, строкой символов, подвыражением, разделенным «|» (обратите внимание, что речь о вертикальной черте, а не заглавной букве «i»)	b(a|i|e)d	bad, bid, bed
.	Ищет совпадения с любым отдельным символом (включая символы, цифры, пробел и т. д.)	b.d	bad, bzd, b$d, b d
^	Указывает на то, что символ или подвыражение расположено в начале строки	^a	apple, asdf, a
\	Символ экранирования (который позволяет вам использовать «специальные» символы в их литеральном значении)	\. | \\	. | \
$	Часто используется в конце регулярного выражения, означает «искать совпадения с этим символом в конце строки». Без него каждое регулярное выражение де-факто заканчивается на «.*», учитывая совпадения лишь с символами, указанными в первой части строки. Он аналогичен символу ^	[A-Z]*[a-z]*$	ABCabc, zzzyx, Bob
?!	«Не содержит». Это странное сочетание вопросительного и восклицательного знаков перед символом (или регулярным выражением) указывает на то, что данный символ должен отсутствовать в заданном месте строки. Его надо использовать аккуратно, например потому, что символ может встретиться в другой части строки. Если вы хотите исключить символ вообще, используйте это сочетание вместе с символами ^ and $ в обоих концах строки	^((?![A-Z]).)*$	Здесь нет заглавных букв, $ymb0ls a4e fine

Регулярные выражения не всегда регулярны!

Стандартные регулярные выражения (о которых мы рассказываем в этой книге и используем в Python и BeautifulSoup) основаны на синтаксисе языка Perl. Большинство современных языков программирования использует его или синтаксис, очень схожий с ним. Однако помните, что при написании регулярных выражений на другом языке вы можете столкнуться с проблемами. Даже некоторые современные языки, например Java, немного отличаются по способам обработки регулярных выражений. Если вы сомневаетесь, обратитесь к документации!

Регулярные выражения и BeautifulSoup

Если предыдущий раздел, посвященный регулярным выражениям, показался вам немного выбивающимся из общей канвы книги, то в этом разделе мы все свяжем воедино. BeautifulSoup и регулярные выражения идут рука об руку, когда нужно применить скрапинг веб-сайтов. На самом деле большинство функций, которые принимают строковый аргумент (например, `find(id="aTagIdHere")`), точно так же принимают регулярное выражение.

Давайте взглянем на некоторые примеры, выполнив скрапинг страницы, расположенной по адресу http://bit.ly/1KGe2Qk.

Обратите внимание, что на сайте размещено большое количество изображений продуктов – они имеют следующий вид:

```
<img src="../img/gifts/img3.jpg">
```

На первый взгляд процесс захвата URL-адресов всех изображений продуктов может показаться довольно простым: просто захватим все теги изображений с помощью `.findAll("img")`, ведь так? Но есть проблема. Кроме очевидных «ненужных» изображений (например, логотипов), современные веб-сайты часто содержат скрытые изображения, прозрачные картинки, используемые для верстки страниц, и другие теги изображений, о существовании которых вы не подозреваете. Конечно, вы не можете рассчитывать на то, что единственными изображениями, размещенными на веб-странице, будут изображения продуктов.

Вдобавок предположим, что макет страницы может измениться, или по какой-то причине, желая найти нужный тег, мы не хотим зависеть от *расположения* картинки на странице. Бывают ситуации, когда вы пытаетесь захватить определенные элементы или куски данных, которые хаотично разбросаны по веб-сайту. Например, в верхней части макета некоторых страниц может располагаться миниатюра изображения продукта.

Решение заключается в том, чтобы найти информацию, идентифицирующую сам тег. В данном случае мы можем посмотреть пути расположения файлов изображений:

```
from urllib.request import urlopen
from bs4 import BeautifulSoup
import re

html = urlopen("http://www.pythonscraping.com/pages/page3.html")
bsObj = BeautifulSoup(html)
images = bsObj.findAll("img", {"src":re.compile("\.\.\/img\/gifts/img.*\.jpg")})
for image in images:
    print(image["src"])
```

Данный код печатает только относительные пути к файлам изображений, которые начинаются с *../img/gifts/img* и заканчиваются на *.jpg*. Вывод показан ниже:

```
../img/gifts/img1.jpg
../img/gifts/img2.jpg
../img/gifts/img3.jpg
../img/gifts/img4.jpg
../img/gifts/img6.jpg
```

Регулярное выражение можно вставить в качестве аргумента в выражение BeautifulSoup, что позволит вам сделать поиск интересующих элементов значительно более гибким.

Работа с атрибутами

До сих пор мы рассказывали о том, как прочитать и отфильтровать теги, а также получить доступ к их содержимому. Однако очень часто в ходе веб-скрапинга вам надо найти не контент, заключенный внутри тега, а атрибуты тега. Особенно это важно для тега `<a>`, который с помощью атрибута `href` устанавливает ссылку на сайт, его отдельную страницу или файл (в качестве ссылки используется URL-адрес), или тега `<img>`, который с помощью атрибута `src` задает адрес файла изображения.

Получить доступ к питоновскому списку атрибутов можно автоматически, вызвав:

```
myTag.attrs
```

Имейте в виду, что эта строка буквально возвращает объект питоновского словаря, который делает поиск и работу с атрибутами три-

виальными операциями. Например, адрес файла изображения можно найти с помощью следующей строки:

```
myImgTag.attrs['src']
```

Лямбда-выражения

Если вы получили образование в сфере информатики, вы, вероятно, слышали о лямбда-выражениях в школе, но впоследствии никогда не использовали. Если нет, они могут быть вам незнакомы (или воспринимаются вами как «нечто, о чем я давно должен был узнать в какой-то момент»). В этом разделе мы не будем углубляться в работу этих необычайно полезных функций, тем не менее мы рассмотрим несколько примеров, где они могут пригодиться для веб-скрапинга.

По сути, лямбда-выражение – это функция, которая передается в другую функцию в качестве переменной. То есть вместо определения функции как f(x, y) вы можете задать функцию как f(g(x), y) или даже f(g(x), h(x)).

BeautifulSoup позволяет передать определенные типы функций в качестве параметров в функцию `findAll`. Единственное ограничение – в том, что эти функции должны принять теговый объект в качестве аргумента и возвратить булево значение. Каждый теговый объект, обрабатываемый BeautifulSoup, оценивается в этой функции, и теги, которые оцениваются как «истинные», возвращаются, в то время как остальные отбрасываются.

Например, следующая строга извлекает все теги, у которых ровно два атрибута:

```
soup.findAll(lambda tag: len(tag.attrs) == 2)
```

То есть она найдет следующие теги:

```
<div class="body" id="content"></div>
<span style="color:red" class="title"></span>
```

Используя лямбда-функции в BeautifulSoup, селекторы могут заменить регулярные выражения, если вам удобнее работать с кодом небольшого объема.

За рамками BeautifulSoup

Несмотря на то что библиотека BeautifulSoup будет использоваться на протяжении всей этой книги (и является одной из самых популяр-

ных HTML-библиотек для Python), имейте в виду, что, помимо нее, есть еще библиотеки для обработки HTML. Если BeautifulSoup не отвечает вашим потребностям, можно обратиться к следующим широко используемым библиотекам:

- lxml: эта библиотека используется для парсинга документов HTML и XML и известна тем, что является низкоуровневой и в значительной мере реализована на C. Хотя потребуется немало времени, чтобы изучить ее (на самом деле крутая кривая обучения указывает на очень быстрое приобретение навыка), она позволяет очень быстро выполнять парсинг большинства документов HTML.
- HTML Parser: это встроенная библиотека Python, предназначенная для парсинга. Поскольку она не требует установки (кроме того что в первую очередь должен быть установлен Python), она чрезвычайно удобна в использовании.

Глава 3

Запуск краулера

До сих примеры, приведенные в книге, касались отдельных статичных страниц с несколько неправдоподобными примерами. В этой главе мы взглянем на некоторые реальные проблемы, рассмотрим работу скраперов, которые способны перемещаться по нескольким страницам и даже нескольким сайтам.

Веб-краулеры, или веб-пауки, называются так потому, что они ползают по вебу. В основе их работы – рекурсивный обход. Они должны извлечь содержимое страницы по указанному URL-адресу, исследовать эту страницу на наличие другого URL-адреса, извлечь страницу по найденному URL-адресу и так до бесконечности.

Однако будьте осторожны: возможность детально просканировать Интернет вовсе не означает, что вы всегда должны прибегать к ней. Скраперы, использованные в предыдущих примерах, отлично работают в ситуациях, когда все нужные данные находятся на одной странице. Используя веб-краулеры, вы должны быть крайне внимательны в настройке пропускной способности и приложить все усилия, чтобы снизить нагрузку на интересующий вас сервер.

Обход отдельного домена

Даже если вы никогда не слышали о «Шести шагах Википедии», вы почти наверняка слышали о «Шести шагах до Кевина Бэйкона». Цель обеих игр заключается в том, чтобы связать двух непохожих субъектов (в первом случае это статьи из Википедии, ссылающиеся друг на друга, во втором случае, актеры, снявшиеся в одном и том же фильме), используя цепочку, состоящую не более чем из шести субъектов (включая два исходных субъекта).

Например, Эрик Айдл снялся в фильме *«Дадли Справедливый»* с Бренданом Фрейзером, который в свою очередь снялся в фильме *«Воздух, которым я дышу»* с Кевином Бэйконом[1].

---

[1] Выражаю признательность создателям сайта Oracle of Bacon за то, что удовлетворили мое любопытство по поводу данной цепочки.

В этом случае цепочка от Эрика Айдла до Кевина Бэйкона состоит лишь из трех субъектов.

В этом разделе мы запустим проект, который станет решением задачи «Шести шагов Википедии». То есть мы можем взять страницу Эрика Айдла в Википедии и вычислить наименьшее число кликов, ведущее нас на страницу Кевина Бэйкона.

Однако как насчет нагрузки на сервер Википедии!

Согласно фонду «Викимедиа» (головной организации Википедии), веб-сайт получает примерно 2500 запросов в секунду, более 99% из них адресованы к домену Википедии (см. раздел «Traffic Volume» на странице «Wikimedia in Figures»). Из-за огромного объема трафика ваши веб-скраперы вряд ли как-то заметно повлияют на нагрузку сервера Википедии. Однако если вы будете активно использовать примеры кода, приведенные в этой книге, или запустите свои собственные проекты по скрапингу Википедии, я призываю вас сделать пожертвование в фонд «Викимедиа» – даже несколько долларов скомпенсируют вашу нагрузку на сервер и помогут сделать данный образовательный ресурс доступным для всех остальных пользователей.

Вы, наверное, уже знаете, как написать скрипт Python, который извлекает произвольную страницу Википедии и выдает список ссылок, расположенных на этой странице:

```
from urllib.request import urlopen
from bs4 import BeautifulSoup

html = urlopen("http://en.wikipedia.org/wiki/Kevin_Bacon")
bsObj = BeautifulSoup(html)
for link in bsObj.findAll("a"):
    if 'href' in link.attrs:
        print(link.attrs['href'])
```

Если вы посмотрите на список полученных ссылок, то заметите, что все статьи, которые вы ожидали увидеть, там перечислены: «Аполлон 13», «Филадельфия», «Премия «Эмми Прайм-тайм» и т. д. Однако также есть и некоторые статьи, которые нам не нужны:

```
//wikimediafoundation.org/wiki/Privacy_policy
//en.wikipedia.org/wiki/Wikipedia:Contact_us
```

На самом деле Википедия пестрит ссылками в боковой панели, подвале и заголовке, которые располагаются на каждой странице, а также ссылками на категории, страницы обсуждений и другие страницы, которые не содержат статей:

```
/wiki/Category:Articles_with_unsourced_statements_from_April_2014
/wiki/Talk:Kevin_Bacon
```

Недавно мой друг, выполняя аналогичный проект по скрапингу Википедии, упомянул, что написал очень большую функцию (более 100 строк кода), чтобы определить, является внутренняя ссылка Википедии страницей-статьей или нет. К сожалению, перед написанием кода он не стал тратить много времени, пытаясь найти закономерности между «статейными ссылками» и «другими ссылками», или он, возможно, обнаружил какую-то хитрую закономерность. Если вы посмотрите на ссылки, ведущие на страницы-статьи (в отличие от других внутренних страниц), все они имеют три общие черты:

- они располагаются внутри блоков `div` с `id="bodycontent"`;
- URL-адреса не содержат точек с запятой;
- URL-адреса начинаются с */wiki/*.

Мы можем использовать эти правила, чтобы немного изменить код для извлечения только нужных статейных ссылок:

```
from urllib.request import urlopen
from bs4 import BeautifulSoup
import re

html = urlopen("http://en.wikipedia.org/wiki/Kevin_Bacon")
bsObj = BeautifulSoup(html)
for link in bsObj.find("div", {"id":"bodyContent"}).findAll("a",
                       href=re.compile("^(/wiki/)((?!:).)*$")):
    if 'href' in link.attrs:
        print(link.attrs['href'])
```

Запустив этот код, вы получите список всех URL-адресов статей, с которыми связана статья Википедии о Кевине Бэйконе.

Конечно, написание скрипта, который найдет все статейные ссылки в одной конкретной статье Википедии, хоть и интересно, но довольно бесполезно на практике. Мы можем взять этот код и превратить его в нечто большее, использовав:

- отдельную функцию `getLinks`, которая берет URL-адрес статьи Википедии в формате `/wiki/<Article_Name>` и возвращает список URL-адресов всех связанных с ней статей в том же самом формате;
- основную функцию, которая вызывает `getLinks` с определенной стартовой статьи, выбирает случайную статейную ссылку из возвращаемого списка и вызывает снова `getLinks`, и так до тех пор, пока мы не остановим программу или пока на новой странице не будет найдено ни одной статейной ссылки.

Вот полный код, который выполняет эту операцию:

```
from urllib.request import urlopen
from bs4 import BeautifulSoup
import datetime
import random
import re

random.seed(datetime.datetime.now())
def getLinks(articleUrl):
    html = urlopen("http://en.wikipedia.org"+articleUrl)
    bsObj = BeautifulSoup(html)
    return bsObj.find("div", {"id":"bodyContent"}).findAll("a",
                    href=re.compile("^(/wiki/)((?!:).)*$"))
links = getLinks("/wiki/Kevin_Bacon")
while len(links) > 0:
    newArticle = links[random.randint(0, len(links)-1)].attrs["href"]
    print(newArticle)
    links = getLinks(newArticle)
```

Первое, что делает программа после импорта необходимых библиотек, – устанавливает начальное значение генератора случайных чисел, используя текущее системное время. Это практически гарантирует уникальный и случайный маршрут перемещения по статьям Википедии при каждом запуске программы.

Стартовые значения для генератора псевдослучайных чисел

В предыдущем примере я использовала генератор случайных чисел Python, чтобы случайно выбрать статью на каждой странице и продолжить случайный обход Википедии. Однако случайные числа нужно использовать с осторожностью.

Несмотря на то что компьютеры сильны в вычислении правильных ответов, у них очень плохо с фантазией. По этой причине применение случайных чисел может стать проблемой. Большинство алгоритмов случайных чисел старается сгенерировать равномерно распределенную и труднопредсказуемую последовательность чисел, но для их запуска требуется сначала задать стартовое значение. Одно и то же стартовое значение будет генерировать каждый раз одну и ту же последовательность случайных чисел. По этой причине я использовала системные часы в качестве стартового значения для получения новых последовательностей случайных чисел и, таким образом, новых последовательностей случайных статей. Это делает запуск программы чуть более интригующим.

Для любознательных сообщаю, что генератор псевдослучайных чисел Python использует алгоритм вихря Мерсенна. Генерируя равномерно распределенные и труднопредсказуемые случайные числа, он требует немного больше процессорных ресурсов. Случайные числа – удовольствие не из дешевых!

Затем она задает функцию `getLinks`, которая берет URL-адрес статьи в формате `/wiki/...`, добавляет доменное имя Википедии

http://en.wikipedia.org и извлекает объект BeautifulSoup из HTML-страницы в этом домене. Потом она извлекает список тегов, содержащих статейные ссылки, используя ранее обсуждавшиеся параметры, и возвращает их.

Основная часть программы начинается с того, что определяет список тегов, содержащих статейные ссылки (переменная `links`), по списку ссылок на начальной странице: http://bit.ly/1KwqjU7. Затем программа делает проход, находит случайный тег статейной ссылки на странице, извлекает из него атрибут *href*, печатает страницу и получает новый список ссылок из извлеченного URL-адреса.

Конечно, на решение проблемы «Шести шагов Википедии» уйдет больше времени, чем на обычное создание скрапера, перемещающегося от страницы к странице. Кроме того, у нас должна быть возможность хранить и анализировать полученные данные. Чтобы посмотреть, как эта проблема будет решена в дальнейшем, обратитесь к главе 5.

Обработайте исключения!

Хотя мы преимущественно опускаем обработку исключений в примерах кода для краткости, имейте в виду, что может возникнуть много подводных камней. Например, что делать, если Википедия изменила имя тега `bodyContent`? (Подсказка: при выполнении кода будет выдана ошибка.) Таким образом, несмотря на то что эти скрипты, возможно, прекрасно запустятся, поскольку тщательно составлены, самостоятельно написанный код потребует более тщательной обработки исключений, чем мы можем рассказать в этой книге. Обратитесь к главе 1 для получения более подробной информации об этом.

Краулинг всего сайта

В предыдущем разделе мы совершили произвольную прогулку по веб-сайту, перемещаясь от ссылки к ссылке. Но что, если вам нужно систематизировать или извлечь все страницы сайта? Краулинг всего сайта, особенно крупного, представляет собой процесс, требующий большого объема памяти, и лучше всего подходит для приложений, использующих базы данных для хранения результатов краулинга. Однако мы можем исследовать работу этих приложений, не запуская их полностью. Для получения более подробной информации о запуске приложений, использующих базы данных, смотрите главу 5.

Темный и глубокий Интернеты

Вы, вероятно, слышали термины *глубокий Интернет (deep Web)*, *темный Интернет (dark Web)*, *скрытый Интернет (hidden Web)*, которые особенно часто используются средствами массовой информации в последнее вре-

мя. Что подразумевается под ними? Глубокий Интернет – это просто часть Интерната, которая не относится к *поверхностному Интернету (surface Web)*, индексирующемуся поисковыми системами. Оценки варьируют, но глубокий Интернет почти наверняка составляет около 90% всего Интернета. Поскольку Google не может самостоятельно отправить формы, найти страницы, которые не связаны с доменом верхнего уровня, или исследовать сайты, где *robots.txt* запрещает делать это, доля поверхностного Интернета остается относительно маленькой.

Темный Интернет, также известный как Даркнет, – это совершенно другое дело. Он запускается в существующей сетевой инфраструктуре, но использует клиент Tor, который работает по протоколу SOCKS, обеспечивая защищенный канал для обмена информацией. Несмотря на то что выполнить скрапинг Даркнета вполне возможно, подобно тому, как вы делаете скапинг любого другого веб-сайта, освещение данной темы выходит за рамки этой книги.

В отличие от темного Интернета, выполнить скрапинг глубокого Интернета относительно легко. На самом деле в этой книге приводится много инструментов, которые научат вас тому, как выполнять поиск и извлечение информации из различных мест, до которых поисковые роботы Google не могут добраться.

Итак, когда краулинг всего сайта может быть полезен и когда он на самом деле не нужен? Веб-скраперы, совершающие обход всего сайта, делают много чего полезного:

- *Создание карты сайта.* Несколько лет назад я столкнулась с проблемой: один важный клиент попросил меня оценить стоимость услуг по редизайну сайта, но не хотел предоставлять мне доступ к внутренней системе управления контентом, и у него не было общедоступной карты сайта. Я использовала краулер для обхода всего сайта, собрала все внутренние ссылки и упорядочила страницы, создав точно такую же структуру папок, какая использовалась на сайте. Это позволило мне быстро найти разделы сайта, о существовании которых я даже не подозревала, и точно подсчитать, сколько страниц потребуется обновить с точки зрения дизайна и какой объем информации нужно будет перенести со старого сайта на обновленный.
- *Сбор данных.* Другой мой клиент хотел собрать статьи (рассказы, посты в блогах, новостные статьи и т. д.), чтобы создать работающий прототип платформы специализированного поиска. Несмотря на то что в исчерпывающем краулинге этих вебсайтов не было нужды, он все же был довольно масштабным (там было всего несколько сайтов, которые нас интересовали в плане сбора данных). Я разработала краулеры, которые рекурсивно обходили каждый сайт и собирали лишь данные, размещенные на страницах-статьях.

Общий подход к исчерпывающему краулингу сайта заключается в том, чтобы начать сканирование со страницы верхнего уровня (например, с главной страницы) и найти список всех внутренних ссылок на этой странице. Все эти ссылки затем сканируются, и в каждой из них находится новый список ссылок, запускается новый цикл сканирования.

Очевидно, что это быстро приведет к взрывному росту количества обрабатываемых страниц. Если каждая страница имеет 10 внутренних ссылок, а глубина веб-сайта имеет 5 уровней (обычная глубина для сайта среднего размера), то количество страниц, необходимое для полного обхода сайта, будет равно 10^5, или 100 000 страниц. Как ни странно, несмотря на то что «5 уровней в глубину и 10 внутренних ссылок на странице» – это довольно стандартные размеры веб-сайта, сайтов с более чем 100 000 страниц очень мало. Разумеется, причина заключается в том, что подавляющее большинство внутренних ссылок дублируется.

Во избежание двойного обхода одной и той же страницы, крайне важно, чтобы в ходе выполнения программы все найденные внутренние ссылки форматировались последовательно и сохранялись в рабочем списке результатов поиска. Лишь ранее не встречавшиеся ссылки должны участвовать в процессе сканирования и поиска новых ссылок:

```
from urllib.request import urlopen
from bs4 import BeautifulSoup
import re

pages = set()
def getLinks(pageUrl):
    global pages
    html = urlopen("http://en.wikipedia.org"+pageUrl)
    bsObj = BeautifulSoup(html)
    for link in bsObj.findAll("a", href=re.compile("^(/wiki/)")):
        if 'href' in link.attrs:
            if link.attrs['href'] not in pages:
                #Мы получили новую страницу
                newPage = link.attrs['href']
                print(newPage)
                pages.add(newPage)
                getLinks(newPage)
getLinks("")
```

Для того чтобы получить полное представление о работе веб-краулера, я расширила определение того, что считать «нужной нам внутренней ссылкой».

Теперь скрапер, вместо того чтобы ограничиться страницами-статьями, ищет все ссылки, которые начинаются с */wiki/*, независимо от того, где они находятся на странице, и независимо от того, содержат ли они двоеточия. Вспомним, что страницы-статьи не содержат двоеточия, однако страницы загрузки файлов, страницы обсуждений и т. п. содержат двоеточия в URL-адресе.

Первоначально `getLinks` вызывается с пустым URL в качестве аргумента. Речь идет о «главной странице Википедии»: внутри функции пустой URL-адрес дописывается `http://en.wikipedia.org`. Затем каждая ссылка на первой странице перебирается, и происходит проверка, входит ли эта ссылка в *глобальный набор страниц* (набор страниц, которые скрипт уже определил). Если нет, то она добавляется в список, печатается, и для нее рекурсивно вызывается функция `getLinks`.

Предупреждение относительно рекурсии

В книгах по программному обеспечению это предупреждение редко встретишь, но я подумала, что вы должны знать: если запущенная программа выполняется достаточно долго, она почти наверняка выдаст ошибку.

В Python лимит рекурсии (сколько раз программа может рекурсивно вызвать себя) по умолчанию равен 1000. Поскольку сеть ссылок Википедии является чрезвычайно большой, программа в конечном итоге достигнет лимита рекурсии и остановится, если вы не установите *значение счетчика* рекурсии или что-то, позволяющее предотвратить преждевременную остановку программы.

На «плоских» сайтах, содержащих менее 1000 ссылок, этот метод, как правило, работает очень хорошо, за некоторыми исключениями. Например, я однажды столкнулась с сайтом, где использовалось правило для генерирования внутренних ссылок к постам блогов. Правило звучало так: «взять URL-адрес текущей страницы, на которой мы находимся, и добавить к нему */blog/title_of_blog.php*».

Проблема заключалась в том, что происходило добавление */blog/title_of_blog.php* к URL-адресам, которые *уже* были размещены на странице, содержащей */blog/* в URL-адресе. Таким образом, сайт просто добавлял еще один */blog/*. В конце концов, мой краулер должен был перемещаться по URL-адресам типа */blog/blog/blog/blog.../blog/title_of_blog.php*.

В итоге мне пришлось добавить проверку, чтобы убедиться, что URL-адреса не выглядят смешными, содержащими повторяющиеся сегменты, что могло указывать на бесконечный цикл. Тем не менее, если бы я оставила выполнение этой программы на ночь, не проверив ее, она с легкостью выдала бы ошибку.

Сбор данных по всему сайту

Конечно, работа веб-краулеров была бы довольно скучной, если бы все они просто перепрыгивали с одной страницы на другую. Для того чтобы сделать их работу полезной, мы должны выполнить определен-

ную операцию со страницей, пока мы на ней находимся. Давайте посмотрим, как построить скрапер, который соберет название страницы, первый параграф и ссылку для редактирования страницы (если имеется).

Как всегда, первый шаг, позволяющий определить, как лучше всего выполнить нашу задачу – взглянуть на несколько страниц сайта и задать шаблон. Когда вы проанализируете кучу страниц Википедии (как страницы-статьи, так и страницы без статей, например страницу, на которой прописана политика конфиденциальности), вам станут ясны следующие вещи:

- все заголовки (на каждой странице, независимо от того, является она страницей-статьей, страницей истории изменений или любой другой страницей) проставляются в соответствии с `h1 → span`, и они являются единственными тегами `h1` на странице;
- как упоминалось ранее, текст тела веб-страницы помещается в тег `div# bodyContent`. Тем не менее, если мы хотим конкретизировать выбор и получить только первый параграф текста, нам лучше использовать `div#mw-content-text → p` (выбираем лишь тег первого параграфа). Этот прием можно использовать для всех страниц, содержащих контент, за исключением страниц файлов (например: http://bit.ly/1KwqJtE), в которых нет контента, разбитого на разделы;
- ссылки для редактирования располагаются лишь на страницах-статьях. Если они там есть, они будут найдены в теге `li#ca-edit` в соответствии с `li#ca-edit → span → a`.

Изменив наш базовый код для краулинга, мы можем создать программу, сочетающую краулинг и сбор данных (или, по крайней мере, печать данных):

```
from urllib.request import urlopen
from bs4 import BeautifulSoup
import re

pages = set()
def getLinks(pageUrl):
   global pages
    html = urlopen("http://en.wikipedia.org"+pageUrl)
    bsObj = BeautifulSoup(html)
    try:
        print(bsObj.h1.get_text())
        print(bsObj.find(id ="mw-content-text").findAll("p")[0])
        print(bsObj.find(id="ca-edit").find("span").find("a").attrs['href'])
    except AttributeError:
```

```
        print("This page is missing something! No worries though!")

    for link in bsObj.findAll("a", href=re.compile("^(/wiki/)")):
        if 'href' in link.attrs:
            if link.attrs['href'] not in pages:
                #Мы получили новую страницу
                newPage = link.attrs['href']
                print("----------------\n"+newPage)
                pages.add(newPage)
                getLinks(newPage)
getLinks("")
```

По сути, цикл for в этой программе идентичен циклу for, использованному в исходной программе краулинга (с добавлением символов тире, отделяющих напечатанный контент для лучшей читаемости).

Поскольку мы никогда не знаем, как располагаются данные на страницах сайта, каждый оператор печати расположен в такой последовательности, которая наиболее вероятна для данного сайта. Тег заголовка <h1> размещается на каждой странице (во всяком случае, насколько я могу судить), поэтому мы сперва попытаемся получить эти данные. Затем мы извлекаем текстовый контент, который содержится в большинстве страниц (за исключением страниц файлов). Кнопка «Edit» размещается только на тех страницах, которые уже содержат как заголовки, так и текстовый контент, но она есть не на всех этих страницах.

Различные шаблоны для различных задач

Очевидно, что существуют некоторые опасности, связанные с переносом строк в обработчике исключений. Во-первых, вы не можете сказать, в какой строке произошло исключение. Кроме того, если по некоторым причинам страница содержала кнопку «Edit» без заголовка, то кнопка «Edit» не будет отражена в логе. Однако зачастую этого достаточно, особенно когда порядок появления элементов предсказуем и непреднамеренный пропуск в логах не является большой проблемой.

Вы могли заметить, что в этом и предыдущих примерах мы не столько «собирали» данные, сколько «печатали» их. Очевидно, что данными в терминале довольно трудно управлять. Вопросы хранения информации и создания баз данных мы подробнее рассмотрим в главе 5.

Краулинг Интернета

Всякий раз, когда я рассказываю о веб-скрапинге, кто-нибудь да обязательно спросит: «Как создать Google?» Мой ответ всегда касается двух моментов: «Сначала тратите миллиарды долларов на покупку

самых больших хранилищ данных в мире и прячете их в различных местах по всему миру. Затем создаете веб-краулер».

Когда компания Google начинала свою работу в 1994 году, она состояла всего их двух аспирантов Стэнфордского университета со старым сервером и веб-краулером, написанном на Python. Теперь, когда вы знаете это, у вас есть все необходимые инструменты, которые помогут вам стать следующим мультимиллиардером, создавшим свой капитал в области информационных технологий.

Если серьезно, веб-краулеры – это сердцевина всего того, что приводит в действие многие современные веб-технологии, и вам не обязательно нужно огромное хранилище данных, чтобы использовать их. Для того чтобы провести междоменный анализ данных, вам нужно создать краулеры, которые смогут интерпретировать и хранить данные, собранные с бесчетного количества интернет-страниц.

Так же, как и в предыдущем примере, веб-скаулеры, которые мы собираемся разработать, будут следовать по ссылкам от страницы к странице, создавая карту Интернета. Но на этот раз они не будут игнорировать внешние ссылки, они будут следовать по ним. В ходе перемещения по страницам мы сможем записывать определенную информацию о каждой странице для решения возникших проблем. Краулинг Интернета выполнить сложнее, чем краулинг отдельного домена, который мы делали раньше, потому что различные веб-сайты имеют совершенно разную структуру. Это означает, что мы должны очень аккуратно определять тип интересующей нас информации и способ ее поиска.

Впереди – незнакомые воды

Имейте в виду, что код, приведенный в следующем разделе, может проследовать *куда угодно* в Интернете. Теперь, узнав о «Шести шагах Википедии», вы понимаете, что вполне возможно перейти с сайта типа http://www.sesamestreet.org/ на что-то, менее привлекательное, в несколько прыжков.

Дети, прежде чем запускать этот код, спросите разрешения у ваших родителей. Если у вас чувствительная психика или ваши религиозные убеждения не позволяют читать текст пошлого сайта, читайте примеры кода, но будьте осторожны, запуская их.

Прежде чем начать писать краулер, который волей-неволей просто следует по всем исходящим ссылкам, вы должны задать себе несколько вопросов:

- Какие данные я пытаюсь собрать? Можно ли их собрать, выполнив скрапинг всего лишь нескольких заранее определенных веб-сайтов (почти всегда это более легкий вариант), или

же мой краулер должен искать новые веб-сайты, о существовании которых я, возможно, не знаю?

- Когда мой краулер достигнет конкретного сайта, он сразу пройдет по следующей исходящей ссылке к новому веб-сайту или будет некоторое время бродить по найденному сайту и выполнять его углубленное сканирование?
- Есть ли какие-либо условия, которые делают скрапинг конкретного сайта ненужным? Мне интересен контент не на английском языке?
- Как я смогу защитить себя от судебных исков, если мой краулер привлечет внимание веб-мастера на одном из сканируемых сайтов? (Ознакомьтесь с приложением С для получения дополнительной информации по этому вопросу.)

Гибкий набор функций Python, которые можно сочетать друг с другом для выполнения различных видов веб-скрапинга, можно легко уместить менее чем в 50 строк кода:

```
from urllib.request import urlopen
from bs4 import BeautifulSoup
import re
import datetime
import random

pages = set()
random.seed(datetime.datetime.now())

#Извлекает список всех внутренних ссылок, найденных на странице
def getInternalLinks(bsObj, includeUrl):
    internalLinks = []
    #Находит все ссылки, которые начинаются с "/"
    for link in bsObj.findAll("a", href=re.compile("^(/|.*"+includeUrl+")")):
        if link.attrs['href'] is not None:
            if link.attrs['href'] not in internalLinks:
                internalLinks.append(link.attrs['href'])
    return internalLinks

#Извлекает список всех внешних ссылок, найденных на странице
def getExternalLinks(bsObj, excludeUrl):
    externalLinks = []
    #Находит все ссылки, которые начинаются с "http" или "www"
    #не содержат текущего URL-адреса
    for link in bsObj.findAll("a",
                              href=re.compile("^(http|www)((?!"+excludeUrl+").)*$")):
        if link.attrs['href'] is not None:
            if link.attrs['href'] not in externalLinks:
                externalLinks.append(link.attrs['href'])
```

```
    return externalLinks

def splitAddress(address):
    addressParts = address.replace("http://", "").split("/")
    return addressParts

def getRandomExternalLink(startingPage):
    html = urlopen(startingPage)
    bsObj = BeautifulSoup(html)
    externalLinks = getExternalLinks(bsObj, splitAddress(startingPage)[0])
    if len(externalLinks) == 0:
        internalLinks = getInternalLinks(startingPage)
        return getNextExternalLink(internalLinks[random.randint(0,
                                   len(internalLinks)-1)])
    else:
        return externalLinks[random.randint(0, len(externalLinks)-1)]

def followExternalOnly(startingSite):
    externalLink = getRandomExternalLink("http://oreilly.com")
    print("Random external link is: "+externalLink)
    followExternalOnly(externalLink)

followExternalOnly("http://oreilly.com")
```

Программа, приведенная выше, запускается с http://oreilly.com и случайным образом скачкообразно перемещается от одной внешней ссылки к другой. Ниже приводится пример вывода, который генерирует эта программа:

```
Random external link is: http://igniteshow.com/
Random external link is: http://feeds.feedburner.com/oreilly/news
Random external link is: http://hire.jobvite.com/CompanyJobs/Careers.aspx?c=q319
Random external link is: http://makerfaire.com/
```

Не всегда есть гарантия, что внешние ссылки будут найдены на первой странице сайта. Для того чтобы найти внешние ссылки в этом случае, применяется метод, аналогичный тому, что использовался в предыдущем примере краулинга. Он выполняет рекурсивное сканирование веб-сайта до тех пор, пока не будет найдена внешняя ссылка.

Рисунок 3.1 визуализирует данную операцию в виде блок-схемы.

Примеры программ не предназначены для рабочего внедрения

Я повторюсь, примеры программ, приведенные в этой книге, адаптированы в целях экономии места и удобочитаемости. Не все из них прошли проверку и обработку исключений, необходимую для полнофункционального рабочего кода.

Например, если на сайте, с которым работает краулер, не найдено ни одной внешней ссылки (маловероятно, но это должно произойти в какой-

то момент), программа будет выполняться до тех пор, пока не будет достигнут лимит рекурсии.
Перед запуском кода убедитесь, что вы выполнили все необходимые проверки для обхода возможных подводных камней.

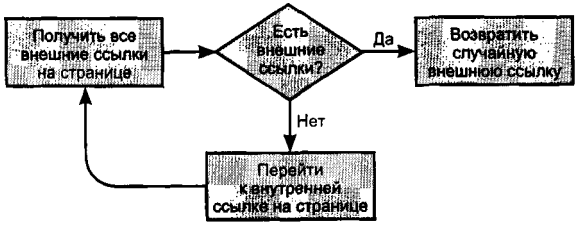


Рис. 3.1 ❖ Блок-схема для скрипта, который перемещается по различным сайтам в Интернете

Преимущество разбивки задач на простые функции типа «найти все внешние ссылки на этой странице» заключается в том, что код можно затем легко модифицировать для выполнения другой задачи краулинга. Например, если наша цель заключается в том, чтобы просканировать весь сайт на наличие внешних ссылок и зафиксировать каждую из них, мы можем добавить следующую функцию:

```
#Извлекает список всех внешних URL-адресов, найденных на сайте
allExtLinks = set()
allIntLinks = set()
def getAllExternalLinks(siteUrl):
    html = urlopen(siteUrl)
    bsObj = BeautifulSoup(html)
    internalLinks = getInternalLinks(bsObj,splitAddress(siteUrl)[0])
    externalLinks = getExternalLinks(bsObj,splitAddress(siteUrl)[0])
    for link in externalLinks:
        if link not in allExtLinks:
            allExtLinks.add(link)
            print(link)
    for link in internalLinks:
        if link not in allIntLinks:
            print("About to get link: "+link)
            allIntLinks.add(link)
            getAllExternalLinks(link)

getAllExternalLinks("http://oreilly.com")
```

Этот код можно рассматривать как два цикла (один собирает внутренние ссылки, другой собирает внешние ссылки), которые работают в одной связке друг с другом. Блок-схема приведена на рис. 3.2:

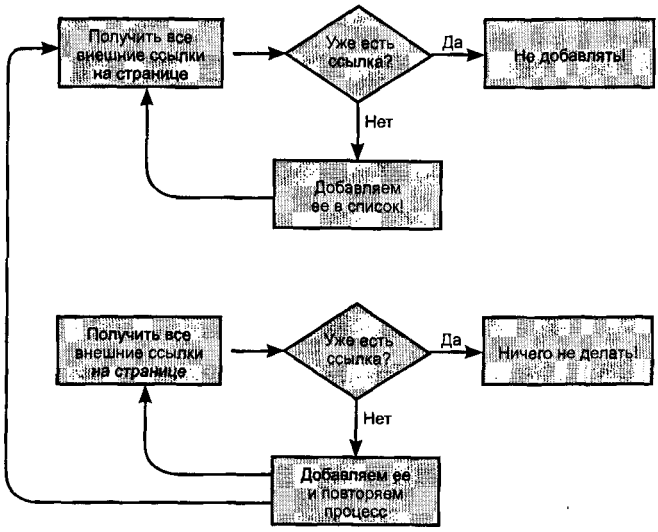


Рис. 3.2 ❖ Блок-схема для краулера сайта, собирающего все внешние ссылки

Составление пояснений или построение диаграммы, иллюстрирующей работу кода, перед написанием самого кода – замечательная привычка, которая поможет вам сэкономить уйму времени и уберечь от совершения ошибок по мере усложнения ваших краулеров.

Обработка редиректов

Редиректы (перенаправления) позволяют просматривать одну и ту же веб-страницу под разными доменными именами. Редиректы бывают двух видов:

- серверные редиректы, когда URL-адрес изменяется перед загрузкой страницы;
- клиентские редиректы, иногда сопровождаемые сообщением типа «Вы будете перенаправлены через 10 секунд...», когда страница загружается перед тем, как произойдет перенаправление на новую страницу.

В этом разделе мы будем работать с серверными редиректами. Для получения более подробной информации о клиентских редиректах, выполняемых с помощью JavaScript или HTML, обратитесь к главе 10.

Что касается серверных редиректов, то вам, как правило, не придется беспокоиться. Если вы используете библиотеку `urllib` и Python 3.x, она обрабатывает редиректы автоматически! Просто надо знать, что иногда URL-адрес страницы, которую вы сканируете, не обязательно является URL-адресом, с помощью которого вы зашли на эту страницу.

Краулинг с помощью Scrapy

Одна из проблем, возникающая при написании веб-краулеров, заключается в том, что вам часто нужно выполнять одни и те же задачи снова и снова: найти все ссылки на странице, отделить внутренние ссылки от внешних, перейти к новым страницам. Эти базовые операции очень важны, вы должны уметь писать их с нуля, однако есть опции, которые понадобятся вам для более детальной обработки страниц.

Scrapy – библиотека Python, которая с легкостью выполняет сложный поиск и обработку ссылок на сайте, осуществляет краулинг доменов или списков доменов. К сожалению, Scrapy еще не выпущена для Python 3.x, хотя она совместима с Python 2.7.

Хорошей новостью является то, что различные версии Python (например, Python 2.7 и 3.4), как правило, работают хорошо, будучи установленными на одной и той же машине. Если вы хотите использовать библиотеку Scrapy для проекта, но при этом хотите применять различные скрипты, предназначенные для Python 3.4, их одновременное использование не станет проблемой.

Веб-сайт Scrapy предлагает загрузить одноименный инструмент, а также инструкции по установке Scrapy с помощью менеджеров-инсталляторов, например таких, как *pip*. Имейте в виду, что вам нужно установить Scrapy, используя Python 2.7 (она не совместима с версиями 2.6 или 3.x), и запускать все программы, используя Scrapy и Python 2.7.

Хотя написание краулеров, использующих Scrapy, сравнительно легко, существуют некоторые настройки, которые нужно выполнить для каждого краулера. Чтобы создать новый проект Scrapy в текущем каталоге, запустите из командной строки:

```
$scrapy startproject wikiSpider
```

`wikiSpider` – это имя нашего нового проекта. Он создает новый каталог в каталоге проекта под названием `wikiSpider`. Внутри этого каталога – следующая файловая структура:

- scrapy.cfg
  - wikiSpider
    - __init.py__
    - items.py
    - pipelines.py
    - settings.py
  - spiders
  - __init.py__

Чтобы создать краулер, мы добавим в *wikiSpider/wikiSpider/spiders/articleSpider.py* новый файл *items.py*. Кроме того, в файле *items.py* мы зададим новый элемент `Article`.

Файл *items.py* нужно отредактировать так, чтобы он выглядел следующим образом (комментарии, сгенерированные Scrapy, здесь сохранены, хотя вы можете свободно удалить их):

```
# -*- coding: utf-8 -*-
# Определяем здесь настройки для ваших элементов
#
# Смотрим документацию по адресу:
# http://doc.scrapy.org/en/latest/topics/items.html

from scrapy import Item, Field

class Article(Item):
    # задаем здесь поля для Вашего элемента, как показано ниже:
    # name = scrapy.Field()
    title = Field()
```

Каждый объект `Item` представляет собой отдельную страницу вебсайта. Очевидно, что вы можете задать любое количество полей (`url`, `content`, `header image` и т. д.), но сейчас я просто извлеку поле `title` из каждой страницы.

Во вновь созданном файле `articleSpider.py` напишите следующее:

```
from scrapy.selector import Selector
from scrapy import Spider
from wikiSpider.items import Article


class ArticleSpider(Spider):
    name="article"
    allowed_domains = ["en.wikipedia.org"]
    start_urls = ["http://en.wikipedia.org/wiki/Main_Page",
              "http://en.wikipedia.org/wiki/Python_%28programming_language%29"]

    def parse(self, response):
        item = Article()
        title = response.xpath('//h1/text()')[0].extract()
        print("Title is: "+title)
        item['title'] = title
        return item
```

Название этого объекта (`ArticleSpider`) отличается от имени каталога (*WikiSpider*), указывая на то, что данный класс, в частности, отвечает за обход лишь страниц-статей в рамках более широкой категории WikiSpider. Для работы с большими сайтами, где размещен

разнообразный контент, вы можете задать отдельные элементы для каждого типа контента (посты в блогах, пресс-релизы, статьи и т. п.), каждый со своими полями, но все они должны использоваться в рамках одного и того же проекта.

Вы можете запустить объект `ArticleSpider` из основного каталога *WikiSpider*, набрав:

```
$ scrapy crawl article
```

Эта строка вызывает скрапер по имени элемента `article` (не по имени класса или файла, а по имени, заданном в строке `name="article"` в `ArticleSpider`).

Наряду с некоторой отладочной информацией этот код печатает строки:

```
Title is: Main Page
Title is: Python (programming language)
```

Скрапер проходит по двум страницам, указанным в качестве `start_urls`, собирает информацию и затем завершает свою работу. Не очень-то много для краулера, но применение Scrapy может оказаться полезным, если у вас есть список URL-адресов, для которых нужно выполнить скрапинг. Чтобы задействовать ее в полноценном краулере, необходимо определить набор правил, которые Scrapy будет использовать для поиска новых URL-адресов на каждой сканируемой странице:

```
from scrapy.contrib.spiders import CrawlSpider, Rule
from wikiSpider.items import Article
from scrapy.contrib.linkextractors.sgml import SgmlLinkExtractor

class ArticleSpider(CrawlSpider):
    name="article"
    allowed_domains = ["en.wikipedia.org"]
    start_urls = ["http://en.wikipedia.org/wiki/Python_
                  %28programming_language%29"]
    rules = [Rule(SgmlLinkExtractor(allow=('(/wiki/)((?!:).)*$'),),
                                    callback="parse_item", follow=True)]

    def parse_item(self, response):
        item = Article()
        title = response.xpath('//h1/text()')[0].extract()
        print("Title is: "+title)
        item['title'] = title
        return item
```

Этот краулер запускается из командной строки точно таким же образом, как и предыдущий, но будет выполняться до тех пор, пока его не остановят командой **Ctrl+C** или закрытием окна терминала.

Создание логов с помощью Scrapy

Отладочная информация, сгенерированная библиотекой Scrapy, может быть полезной, но часто является слишком подробной. Вы можете легко настроить уровень логирования, добавив строку в файл *settings.py* в вашем проекте:

```
LOG_LEVEL = 'ERROR'
```

В библиотеке Scrapy существует пять уровней логирования, перечисленных ниже:

- CRITICAL
- ERROR
- WARNING
- DEBUG
- INFO

Если установлен уровень логирования ERROR, будут выведены только логи CRITICAL и ERROR. Если установлен уровень логирования INFO, будут выведены все логи и т. д.

Для вывода содержимого логов в отдельный лог файл вместо терминала просто задайте лог-файл при запуске из командной строки:

```
$ scrapy crawl article -s LOG_FILE=wiki.log
```

Эта строка создает новый лог файл (если его еще нет) в вашем текущем каталоге, выводит все логи и операторы в него.

Scrapy использует объекты `Item`, чтобы определить, какие куски информации нужно извлечь с посещенных страниц. Scrapy может сохранить эту информацию различными способами, например в виде файлов CSV, JSON и XML, используя следующие команды:

```
$ scrapy crawl article -o articles.csv -t csv
$ scrapy crawl article -o articles.json -t json
$ scrapy crawl article -o articles.xml -t xml
```

Разумеется, можно использовать объекты типа `Item` самостоятельно и записать их в файл или базу данных любым нужным вам способом, просто добавив в краулер соответствующий код для функции парсинга.

Scrapy – мощный инструмент, который справляется с различными проблемами, касающимися краулинга Интернета. Он автоматически собирает все URL-адреса и сравнивает их с заранее определенными правилами, проверяет уникальность всех URL-адресов, нормализует относительные URL-адреса, если это необходимо, и рекурсивно сканирует страницы в глубину.

Этот раздел лишь едва затронул возможности библиотеки Scrapy, и я призываю вас обратиться к документации по библиотеке Scrapy или

другим доступным онлайн-ресурсам. Scrapy – очень большая и расширяемая библиотека с большим количеством настроек. Если вы хотели с помощью Scrapy решить какую-то задачу, не упомянутую здесь, помните, что наверняка есть способ (или даже несколько) выполнить ее.

Глава 4

Использование API

Как и у многих программистов, работавших над большими проектами, у меня есть свои страшилки, когда мне приходилось работать с чужим программным кодом. Из-за проблем загромождения глобального пространства имен, проблем ввода, неправильной интерпретации вывода функции получение информации из точки А с целью передачи в метод В могло стать кошмаром.

Именно здесь могут пригодиться интерфейсы прикладного программирования (application programming interfaces, или API): они предлагают великолепные, удобные средства для обмена информацией между несколькими различными приложениями. Не имеет значения, что приложения написаны разными программистами, с использованием различных архитектур или даже на разных языках – API выступают в качестве общепринятого языка для различных программ, которые должны обмениваться информацией друг с другом.

Хотя для различных программ существуют разнообразные API, в последнее время «API» обычно понимается в значении «API веб-приложения». Как правило, программист выполняет запрос к API с помощью HTTP, чтобы получить некоторые типы данных, а API возвращает эти данные в виде XML или JSON. Хотя большинство API по-прежнему поддерживает XML, JSON набирает популярность в качестве протокола кодирования.

Использование преимуществ уже готовой программы для получения предварительно упакованной информации в удобном формате может показаться вам отклонением от остальной части книги. Что ж, это одновременно так и не так. Несмотря на то что большинство специалистов, как правило, не считают применение API веб-скрапингом, оба способа используют те же самые методы (отправка HTTP-запросов) и получают аналогичные результаты (сбор информации), они часто могут в значительной мере дополнять друг друга.

Например, вы, возможно, захотите объединить информацию, собранную веб-скрапером, с информацией, полученной с помощью API, чтобы повысить ее полезность. Далее в этой главе в качестве примера

мы рассмотрим объединение историй правок Википедии (которые содержат IP-адреса) с информацией API, который определяет IP-адреса, чтобы получить представление о географическом местоположении пользователей, принимающих участие в редактировании Википедии по всему миру.

В этой главе мы дадим общий обзор API и принципов их работы, рассмотрим несколько популярных API, использующихся в настоящий момент, и расскажем о том, как вы можете использовать API в ваших собственных веб-скраперах.

Как работают API

Несмотря на то что API не так широко распространены, как должны бы (что было большой мотивацией для написания этой книги, потому что если нет возможности воспользоваться API, можно все равно получить данные с помощью скрапинга), сейчас можно встретить API, позволяющие получить различные виды информации. Интересуетесь музыкой? С помощью различных API вы найдете песни, исполнителей, альбомы и даже информацию о музыкальных стилях и связанных с ними исполнителях. Нужны данные о спорте? Спортивный телевизионный ESPN предлагает API для получения информации о спортсменах, результатах соревнований и т. д. Google на своей странице Google Developers Console предлагает десятки интерфейсов для автоматических переводов, аналитики, геолокации и многого другого.

API чрезвычайно просты в использовании. В самом деле, вы можете отправить простой API-запрос, лишь введя следующую строчку в вашем браузере[1]:

```
http://freegeoip.net/json/50.78.253.58
```

Эта строка выдает следующий ответ:

```
{"ip":"50.78.253.58","country_code":"US","country_name":"United States,"region_
code":"MA","region_name":"Massachusetts","city":"Chelmsford","zipcode":"01824",
"latitude":42.5879,"longitude":-71.3498,"metro_code":"506","area_code":"978"}
```

Итак, вы перешли к веб-адресу в окне браузера, и он выдает какую-то информацию (очень хорошо отформатированную)? Какая разница между API и постоянным сайтом? Несмотря на шумиху вокруг API,

[1] Этот API определяет географическое местоположение по IP-адресу, и я еще воспользуюсь им позже в этой главе. Более подробную информацию можно получить на сайте http://freegeoip.net.

ответ часто звучит так: «разница небольшая». API функционируют на основе HTTP, того же самого протокола, используемого для получения данных веб-сайтов, скачивания файлов да вообще практически для всего, что делается в Интернете. Единственное, что делает API API, – это использование жестко стандартизированного синтаксиса и тот факт, что API предоставляют свои данные в формате JSON или XML, а не HTML.

Общепринятые соглашения

В отличие от веб-скраперов, API для обработки запроса и генерирования ответа используют жестко стандартизированный набор правил. Поэтому легко выучить несколько простых базовых правил, которые помогут вам быстро приступить к работе с любым API, при условии что он правильно написан.

Однако имейте в виду, что некоторые API все же немного отклоняются от этих правил, поэтому важно прочитать документацию по интерфейсам, когда будете в первый раз использовать их, независимо от того, насколько хорошо вы знакомы с API в целом.

Методы

Есть четыре способа отправить запрос к веб-серверу с помощью HTTP:

- `GET`
- `POST`
- `PUT`
- `DELETE`

`GET` используется, когда вы посещаете веб-сайт с помощью адресной строки в браузере. `GET` – это метод, который вы используете, когда вызываете *http://freegeoip.net/json/50.78.253.58*. Вы можете представить `GET`-запрос в виде фразы: «Эй, веб-сервер, предоставь мне, пожалуйста, эту информацию».

`POST` используется, когда вы заполняете регистрационную форму или отправляете информацию внутреннему скрипту сервера. Каждый раз, когда вы авторизуетесь на веб-сайте, вы делаете `POST`-запрос с помощью Вашего имени пользователя и (надеюсь) зашифрованного пароля. Когда вы делаете `POST`-запрос с помощью API, вы говорите: «Пожалуйста, сохраните эту информацию в базе данных».

При работе с веб-сайтами `PUT` используется реже, но применяется время от времени в API. `PUT`-запрос нужен для изменения объ-

екта или информации. Чтобы создать нового пользователя, интерфейсу может потребоваться POST-запрос, однако, если вы захотите изменить электронный адрес этого пользователя, потребуется PUT-запрос[1].

DELETE-запрос прост. Он используется для удаления объекта. Например, если я посылаю DELETE-запрос к *http://myapi.com/user/23*, он удалит пользователя с ID 23. Методы DELETE не часто используются в публичных API, которые в первую очередь разрабатывались для распространения информации, а не для того, чтобы случайные пользователи удаляли информацию из их баз данных. Однако, как и случае с методом PUT, о методе DELETE полезно знать.

Несмотря на то что есть еще несколько методов, которые используются протоколом HTTP, эти четыре метода исчерпывают функционал практически любого API, с которым вы когда-либо будете работать.

Аутентификация

Хотя некоторые API не применяют аутентификацию для выполнения операций (то есть любой человек может вызвать API бесплатно, изначально не регистрируясь), многие современные API перед использованием требуют определенной аутентификации.

Некоторые API требуют аутентификации для того, чтобы взимать плату за вызов API, или они хотят предложить какую-то услугу в виде ежемесячной подписки. Другие вводят аутентификацию для того, чтобы лимитировать количество запросов, отправляемых пользователями (ограничить их определенным количеством запросов в секунду, час или сутки), или ограничить для некоторых пользователей доступ к определенным видам информации или типам вызовов API. Другие API, возможно, не вводят ограничения, но они могут собирать информацию о пользователях и совершаемых ими вызовах для использования в маркетинговых целях.

Все методы аутентификации API обычно построены на использовании определенного *токена*, который передается на веб-сервер с каждым вызовом API. Этот токен либо предоставляется пользователю при

[1] В действительности многие API при обновлении информации используют POST-запросы вместо PUT-запросов. Создается ли новый объект или обновляется старый – часто это зависит от того, как составлен сам API-запрос. Однако лучше знать об этом различии, и при работе с популярными API вы часто будете сталкиваться с PUT-запросами.

регистрации (при этом токен может быть постоянным, обычно в приложениях с низким уровнем безопасности, или сменным) и извлекается с сервера, используя комбинацию имени пользователя и пароля.

Например, чтобы вызвать Echo Nest API для получения списка песен группы Guns N'Roses, мы можем использовать:

```
http://developer.echonest.com/api/v4/artist/songs?api_key=<ваш api-ключ>%20
&name=guns%20n%27%20roses&format=json&start=0&results=100
```

Эта строка передает на сервер значение `api_key`, присвоенное мне при регистрации, что позволяет серверу идентифицировать отправителя запроса как Райан Митчелл и отправить ему данные в формате JSON.

Кроме передачи токенов в URL-адрес запроса напрямую, токены также можно передать на сервер с помощью cookie в заголовке запроса. Мы расскажем о заголовках более подробно далее в этой главе, а также в главе 12, однако на простом примере видно, как их можно отправить с помощью пакета `urllib`, знакомого по предыдущим главам:

```
token = "<ваш api-ключ>"
webRequest = urllib.request.Request("http://myapi.com", headers={"token":token})
html = urlopen(webRequest)
```

Ответы

Как вы уже видели на примере FreeGeoIP, приведенном в начале главы, важная особенность API заключается в том, что они возвращают хорошо структурированные ответы. Наиболее распространенными форматами являются *Расширяемый язык разметки или Extensible Markup Language* (XML) и *Представление объектов JavaScript или JavaScript Object Notation* (JSON).

В последние годы формат JSON стал гораздо более популярен, чем XML, по двум основным причинам. Во-первых, файлы JSON, как правило, меньше по размеру, чем файлы XML с тщательно разработанной структурой. Сравните, например, данные в формате XML:

```
<user><firstname>Ryan</firstname><lastname>Mitchell</lastname><username>Kludgist
</username></user>
```

которые укладываются в 98 символов, и те же самые данные в формате JSON:

```
{"user":{"firstname":"Ryan","lastname":"Mitchell","username":"Kludgist"}}
```

которые умещаются всего в 73 символа, на 36% меньше, чем те же самые данные в формате XML.

Конечно, можно возразить, что файл XML можно отформатировать следующим образом:

```
<user firstname="ryan" lastname="mitchell" username="Kludgist"></user>
```

но этот способ считается плохой практикой, потому что он не поддерживает глубокой вложенности данных. Несмотря на это, данные умещаются в 71 символ, т. е. у файла примерно такая же длина, как и у эквивалентного файла JSON.

Еще одна причина, по которой JSON быстро обогнал XML по популярности, – изменения, произошедшие в веб-технологиях. Раньше для получения API сервером повсеместно использовались скрипты, написанные на PHP или .NET. В настоящее время вполне вероятно, что для отправки API-запросов и получения ответов будет использоваться такой фреймворк, как Angular или Backbone. Серверным технологиям не важен формат поступающих данных. Однако JavaScript-библиотекам, например Backbone, обработать формат JSON проще.

Хотя большинство API по-прежнему поддерживают XML-вывод, в этой книге мы будем использовать примеры с использованием формата JSON. Несмотря на это, будет оптимальным ознакомиться с обоими форматами, если вы еще с ними не сталкивались, – вряд ли они перестанут использоваться в ближайшее время.

Вызовы API

Синтаксис вызова API может варьировать, однако есть несколько стандартных наиболее распространенных методов. При извлечении данных с помощью GET-запроса URL-путь описывает интересующие нас данные, тогда как параметры запроса выступают в качестве фильтров или дополнительных запросов, сопряженных с поиском.

Например, в гипотетическом API вы можете отправить запрос, чтобы извлечь все посты пользователя с ID 1234, написанные в течение августа 2014 года:

```
http://socialmediasite.com/users/1234/posts?from=08012014&to=08312014
```

Многие API требуют, чтобы путь содержал информацию о версии API, необходимый формат данных и другие атрибуты. Например, следующая строка возвращает те же самые данные, используя API версии 4 и формат JSON:

```
http://socialmediasite.com/api/v4/json/users/1234/
posts?from=08012014&to=08312014
```

Другие API требуют, чтобы вы указали формат и информацию о версии API в качестве параметра запроса, например:

```
http://socialmediasite.com/users/1234/posts?format=json&from=08012014&
to=08312014
```

Echo Nest

Echo Nest – это фантастический пример компании, которая активно использует в своей работе веб-скраперы. Хотя некоторые музыкальные сервисы, например Pandora, вручную классифицируют и аннотируют музыку, Echo Nest для классификации исполнителей, песен и альбомов использует автоматизированную обработку и информацию, извлеченную из блогов и новостных статей с помощью скраперов.

Еще лучше то, что данный API находится в свободном доступе для некоммерческого использования[1]. Чтобы воспользоваться этим API, вам понадобится ключ, который вы можете получить, перейдя на страницу Create an Account сайта Echo Nest и зарегистрировавшись (для этого введите имя, адрес электронной почты и пароль).

Несколько примеров

Echo Nest API разработан для работы с несколькими основными типами контента: исполнителями, песнями, треками и жанрами. За исключением жанров, все эти типы контента имеют уникальные идентификаторы, которые используются для извлечения информации в различных форматах с помощью вызовов API. Например, если я хочу получить список песен группы *Monty Python*, я делаю следующий вызов, чтобы получить их ID (не забудьте заменить `<ваш api-ключ>` на ваш собственный API-ключ):

```
http://developer.echonest.com/api/v4/artist/search?api_key=<ваш api-
ключ>&name=monty%20python
```

Этот код сгенерирует следующий результат:

```
{"response": {"status": {"version": "4.2", "code": 0, "message": "Suc
cess"}, "artists": [{"id": "AR5HF791187B9ABAF4", "name": "Monty Pytho
```

[1] Смотрите The Echo Nest Licensing, чтобы получить подробную информацию об ограничениях.

```
n"}, {"id": "ARWCIDE13925F19A33", "name": "Monty Python's SPAMALOT"},
 {"id": "ARVPRCC12FE0862033", "name": "Monty Python's Graham Chapman"
}]}}
```

Кроме того, я могу использовать этот ID, чтобы запросить список песен:

```
http://developer.echonest.com/api/v4/artist/songs?api_key=<ваш api- ключ>&id=
AR5HF791187B9ABAF4&format=json&start=0&results=10
```

В ответ получаю несколько хитов группы *Monty Python*, а также их менее известные песни:

```
{"response": {"status": {"version": "4.2", "code": 0, "message": "Success"},
"start": 0, "total": 476, "songs": [{"id": "SORDAUE12AF72AC547", "title":
"Neville Shunt"}, {"id": "SORBMPW13129A9174D", "title": "Classic (Silbury Hill)
(Part 2)"}, {"id": "SOQXAYQ1316771628E", "title": "Famous Person Quiz (The
Final Rip Off Remix)"}, {"id": "SOUMAYZ133EB4E17E8", "title": "Always Look On
The Bright Side Of Life - Monty Python"}, ...]}}
```

Кроме того, я могу сделать отдельный вызов, используя название группы `monty%20python` вместо уникального ID, и получить ту же самую информацию:

```
http://developer.echonest.com/api/v4/artist/songs?api_key=<ваш api-
ключ>2&name=monty%20python&format=json&start=0&results=10
```

Используя тот же самый ID, я могу запросить список похожих исполнителей:

```
http://developer.echonest.com/api/v4/artist/similar?api_key=<ваш api-ключ>&id
=AR5HF791187B9ABAF4&format=json&results=10&start=0
```

Результаты включают в себя других комедийных исполнителей, например Эрика Айдла, который выступал в группе *Monty Python*:

```
{"response": {"status": {"version": "4.2", "code": 0, "message": "Suc
cess"}, "artists": [{"name": "Life of Brian", "id": "ARNZYOS1272BA7FF
38"}, {"name": "Eric Idle", "id": "ARELDIS1187B9ABC79"}, {"name": "Th
e Simpsons", "id": "ARNR4B91187FB5027C"}, {"name": "Tom Lehrer", "id"
: "ARJMYTZ1187FB54669"}, ...]}}
```

Обратите внимание, хотя список схожих исполнителей содержит некоторую по-настоящему интересную информацию (например, «Tom Lehrer»), первым результатом является «The Life of Brian» из саундтрека, выпущенного группой *Monty Python*. Одна из опасностей использования базы данных, собранной из различных источников

с минимальным вмешательством человека, заключается в том, что вы можете получить несколько странные результаты. Это нужно иметь в виду при разработке собственного приложения, использующего данные, полученные с помощью API.

Я описала лишь несколько примеров использования Echo Nest API. Полную документацию можно посмотреть в разделе Echo Nest API Overview.

Echo Nest является спонсором различных хакатонов и проектов по программированию, нацеленных на интеграцию технологий и музыки. Если вам это интересно, Echo Nest Demo Page является хорошим местом для старта.

Twitter

Twitter, как известно, защищает свои API, и это справедливо. Имея более 230 млн активных пользователей и доход более $100 млн в месяц, компания не хочет, чтобы кто-то просто так пришел и заполучил все интересующие его данные.

Ограничения, введенные Twitter (количество запросов, которое разрешено сделать каждому пользователю), бывают двух видов: 15 запросов за 15 минут и 180 запросов за 15 минут, в зависимости от типа вызова. Например, вы можете сделать до 12 запросов в минуту, чтобы получить основную информацию о пользователях Twitter, но лишь один запрос в минуту, чтобы получить списки фолловеров этих пользователей[1].

Приступаем к работе

Помимо ограничений, Twitter имеет более сложную систему авторизации для получения ключей и их использования. Для получения ключа вам, конечно, понадобится учетная запись Twitter. Вы можете относительно легко создать ее на странице регистрации https://twitter.com/signup. Кроме того, вы должны зарегистрировать Ваше «приложение» на сайте Twitter Developers.

После прохождения регистрации вы попадете на страницу, содержащую основную информацию о вашем приложении, включая Consumer Key, или Пользовательский ключ (рис. 4.1):

[1] Чтобы получить полный список ограничений, смотрите https://dev.twitter.com/rest/public/rate-limits.

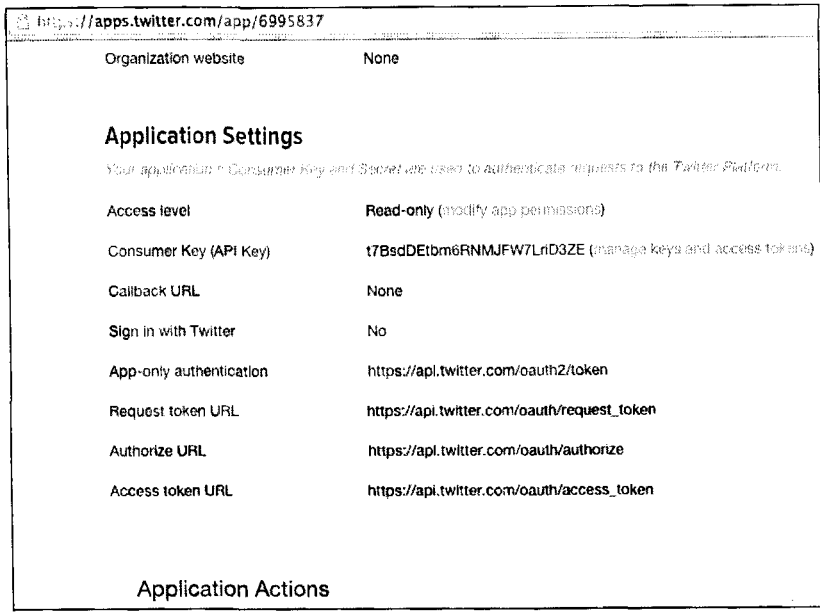


Рис. 4.1 ❖ Страница Application Settings содержит основную информацию о вашем приложении

После щелчка по manage keys and access tokens (управлять ключами и токенами доступа) вы будете направлены на страницу, содержащую дополнительную информацию (рис. 4.2).

Эта страница содержит кнопку для автоматической повторной генерации Consumer Key (Пользовательского ключа) и Consumer Secret (Пользовательского секрета), если они по каким-то причинам стали общедоступными (например, вы случайно опубликовали их в качестве примера в вашей книге по программированию).

Несколько примеров

Система аутентификации Twitter использует протокол OAuth и является довольно сложной. Для работы с этим протоколом лучше использовать одну из доступных библиотек, чем пытаться отслеживать его работу самостоятельно. Из-за относительной сложности работы с Twitter API «в ручном режиме» примеры в этом разделе будут сосредоточены на использовании кода Python для взаимодействия с API, а не на работе с самим API.

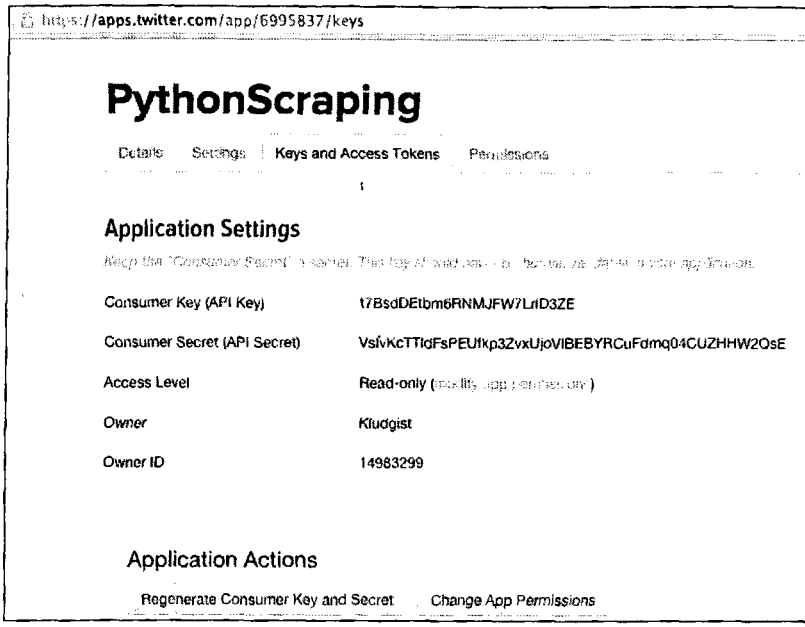


Рис. 4.2 ❖ Для использования Twitter API вам еще нужен Consumer Secret (Пользовательский секрет)

К моменту написания книги вышло довольно много библиотек для Python 2.x, используемых для работы с Twitter, и несколько библиотек для Python 3.x. К счастью, одна из лучших библиотек для работы с Twitter (которая имеет соответствующее название Twitter) доступна для Python 3.x. Вы можете скачать ее со страницы Python Twitter Tools и установить обычным способом:

```
$cd twitter-x.xx.x
$python setup.py install
```

Разрешения на доступ в Twitter

По умолчанию токены доступа, выдаваемые приложениям, имеют разрешение только на чтение. Этого достаточно для выполнения большинства задач, за исключением случаев, когда вы в самом деле хотите, чтобы приложение сделало твит от Вашего имени.

Чтобы изменить разрешение на чтение/запись для токенов, перейдите на вкладку Permissons панели Application Management. Токены нужно будет сгенерировать заново, чтобы изменения вступили в силу.

Аналогично вы можете изменить разрешение, чтобы получить доступ к прямым сообщениям в Twitter, если это необходимо Вашему приложе-

нию. Однако будьте осторожны, вы должны задать разрешения, которые действительно необходимы. В общем случае, лучше создать несколько наборов токенов для нескольких приложений, а не повторно использовать токены, дающие приложениям слишком большие права.

В нашем первом упражнении мы будем искать конкретные твиты. Код, приведенный ниже, подключается к Twitter API и выводит список твитов в формате JSON, содержащий хэш-тег #python. Не забудьте изменить строки в OAuth в соответствии с вашими реальными учетными данными:

```
from twitter import Twitter

t = Twitter(auth=OAuth(<Access Token>,<Access Token Secret>,
                       <Consumer Key>,<Consumer Secret>))
pythonTweets = t.search.tweets(q = "#python")
print(pythonTweets)
```

Вывод этого скрипта может быть огромен, но имейте в виду, что вы получаете развернутую информацию о твите: дата и время публикации твита, подробная информация о ретвитах или избранных твитах, сведения об аккаунте пользователя и изображении профиля. Хотя вам, возможно, понадобится лишь часть этих данных, Twitter API предназначен для веб-разработчиков, которые хотят разместить твиты, полученные с помощью API, на своих собственных веб-сайтах, поэтому выводится много дополнительной информации!

Вы можете увидеть пример вывода одного твита при обновлении статуса с помощью API:

```
from twitter import *

t = Twitter(auth=OAuth(<Access Token>, <Access Token Secret>,
                       <Consumer Key>, <Consumer Secret>))
statusUpdate = t.statuses.update(status='Hello, world!')
print(statusUpdate)
```

Ниже приводится информация о твите в формате JSON:

```
{'created_at': 'Sun Nov 30 07:23:39 +0000 2014', 'place': None, 'in_reply_to_scr
een_name': None, 'id_str': '538956506478428160', 'in_reply_to_user_id: None,'lan
g': 'en', 'in_reply_to_user_id_str': None, 'user': {'profile_sidebar_border_colo
r': '000000', 'profile_background_image_url': 'http://pbs.twimg.com/profile_back
ground_images/497094351076347904/RXn8MUlD.png', 'description':'Software Engine
er@LinkeDrive, Masters student @HarvardEXT, @OlinCollege graduate, writer @OReil
lyMedia. Really tall. Has pink hair. Female, despite the name.','time_zone': 'Ea
stern Time (US & Canada)', 'location': 'Boston, MA', 'lang': 'en', 'url': 'http:
//t.co/FM6dHXloIW', 'profile_location': None, 'name': 'Ryan Mitchell', 'screen_n
```

```
ame': 'Kludgist', 'protected': False, 'default_profile_image': False, 'id_str':
'14983299', 'favourites_count': 140, 'contributors_enabled': False, 'profile_use
_background_image': True, 'profile_background_image_url_https': 'https://pbs.twi
mg.com/profile_background_images/497094351076347904/RXn8MUlD.png', 'profile_side
bar_fill_color': '889654', 'profile_link_color': '0021B3', 'default_profile': Fa
lse, 'statuses_count': 3344, 'profile_background_color': 'FFFFFF', 'profile_imag
e_url': 'http://pbs.twimg.com/profile_images/496692905335984128/XJh_d5f5_normal.
jpeg', 'profile_background_tile': True, 'id': 14983299, 'friends_count': 409, 'p
rofile_image_url_https': 'https://pbs.twimg.com/profile_images/496692905335984128
/XJh_d5f5_normal.jpeg', 'following': False, 'created_at': 'Mon Jun 02 18:35:1
8 +0000 2008', 'is_translator': False, 'geo_enabled': True, 'is_translation_enabl
ed': False, 'follow_request_sent': False, 'followers_count': 2085, 'utc_offset'
: -18000, 'verified': False, 'profile_text_color': '383838', 'notifications': F
alse, 'entities': {'description': {'urls': []}, 'url': {'urls': [{'indices': [
0, 22], 'url': 'http://t.co/FM6dHXloIw', 'expanded_url': 'http://ryanemitchell.
com', 'display_url': 'ryanemitchell.com'}]}}, 'listed_count': 22, 'profile_banne
r_url': 'https://pbs.twimg.com/profile_banners/14983299/1412961553'}, 'retweeted
': False, 'in_reply_to_status_id_str': None, 'source': '<a href="http://ryanemit
chell.com" rel="nofollow">PythonScraping</a>', 'favorite_count': 0, 'text': 'Hell
o,world!', 'truncated': False, 'id': 538956506478428160, 'retweet_count': 0, 'fa
vorited': False, 'in_reply_to_status_id': None, 'geo': None, 'entities': {'user_m
entions': [], 'hashtags': [], 'urls': [], 'symbols': []}, 'coordinates': None, '
contributors': None}
```

Да, это результат отправки одного твита. Иногда я думаю, что Twitter ограничивает доступ к своим API из-за пропускной способности, необходимой для отправки ответа на каждый запрос!

Отправляя запрос, извлекающий список твитов, вы можете ограничить количество полученных твитов, задав определенное значение:

```
pythonStatuses = t.statuses.user_timeline(screen_name="montypython", count=5)
print(pythonStatuses)
```

В данном случае мы запросим последние пять твитов, которые были размещены в ленте @montypython (включая любые сделанные ретвиты).

Хотя эти три примера (поиск твитов, получение твитов определенного пользователя и размещение собственных твитов) охватывают бóльшую часть задач, выполняемых с помощью Twitter API, возможности питоновской библиотеки Twitter гораздо шире. Вы можете получить списки Twitter, управлять ими, зафолловить или расфолловить пользователей, посмотреть информацию профилей пользователей и многое другое. Полную документацию можно найти на GitHub.

Google API

На сегодняшний день Google предлагает наиболее универсальные и простые в использовании API в Интернете. Всякий раз, когда вы сталкиваетесь с какой-то задачей, будь то перевод языка, геолокация, определение времени и даты или даже геномика, Google предлагает для ее решения свой API. Для многих своих популярных приложений (Gmail, YouTube, Blogger) Google также разработал API .

Чтобы составить представление о Google API, можно обратиться к двум главным страницам. Первая страница – это страница Products, которая выступает в качестве структурированного репозитория интерфейсов, комплектов средств разработки программного обеспечения и других проектов, которые могут представлять интерес для разработчиков программного обеспечения. Другая страница – это Google Developers Console, которая предлагает удобный интерфейс для применения API-сервисов, здесь же можно ознакомиться с ограничениями и правилами использования и даже запустить облачные вычисления Google, если вы разбираетесь в этом.

Большинство интерфейсов Google бесплатно, хотя некоторые, например Google Search API, являются платными. Google довольно либерален, предлагая набор бесплатных API, позволяющих отправлять с базового аккаунта от 250 запросов в день до 20 000 000 запросов в день. Для некоторых API предусмотрена возможность увеличить количество обрабатываемых запросов. Вам нужно пройти процедуру идентификации личности с помощью кредитной карты (без взимания денег с карты). Например, Google Places API имеет базовое ограничение в 1000 запросов за 24-часовой период, но его можно повысить до 150 000 запросов, пройдя процедуру идентификации личности. Для получения дополнительной информации смотрите страницу Usage Limits and Billing.

Приступаем к работе

Если у вас уже есть аккаунт Google, можно просмотреть список доступных API и получить API-ключ, используя Google Developers Console. Если у вас нет аккаунта Google, создайте его, воспользовавшись страницей Create Your Google Account.

Войдя в аккаунт или создав его, вы можете посмотреть свои учетные данные, включая API-ключи на странице консоли API, щелкните Credentials (Учетные данные) в меню слева (рис. 4.3):

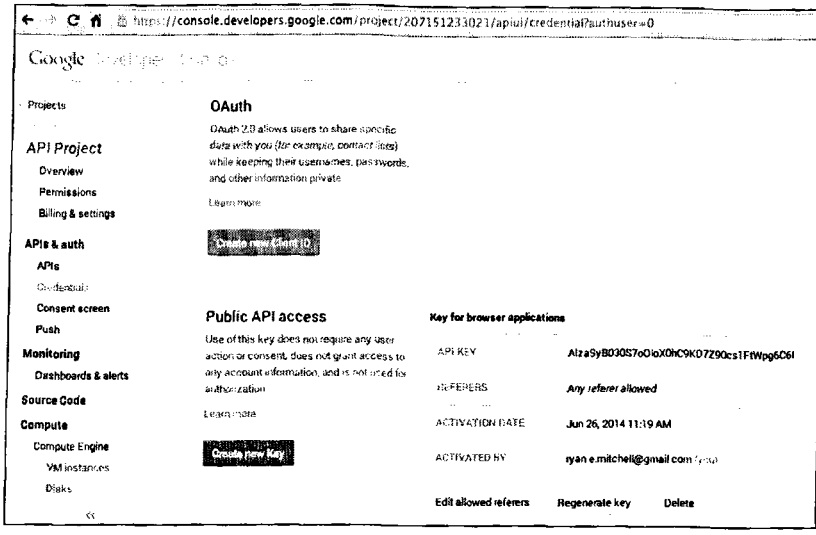


Рис. 4.3 ❖ Страница учетных данных для Google API

На странице **Credentials** (Учетные данные) вы можете щелкнуть по кнопке **Create new Key** (Создать новый ключ) с возможностью разрешить доступ к данному API только с указанного IP-адреса или при переходе с указанного URL-адреса. Чтобы создать API-ключ, который можно использовать для любого URL-адреса или IP-адреса, просто оставьте поле **Accept Request From These Server IP Addresses** (Принимать запросы только от указанных IP-адресов) пустым. Однако имейте в виду, что важно хранить ваш ключ в тайне, если вы не ограничиваетесь исходным IP-адресом, – любые вызовы, сделанные с помощью API-ключа, будут засчитываться, даже если они не авторизированы.

Кроме того, вы можете создать несколько API-ключей. Например, у вас может быть отдельный API-ключ для каждого проекта или веб-домена, которым вы управляете. Однако ограничения Google API, касающиеся количества обрабатываемых запросов, применяются на уровне аккаунта, а не ключа. Поэтому хотя использование нескольких ключей – это удобный способ настроить работу с отдельными API, он не поможет обойти ограничения!

Несколько примеров

Наиболее популярные (и, на мой взгляд, наиболее интересные) интерфейсы можно найти в коллекции Google Maps API. Вы, возможно,

уже знакомы с этим инструментом, используя встроенные Google-карты на различных веб-сайтах. Однако Maps API выходят далеко за рамки возможностей встроенных карт – вы можете определить географические координаты адреса (широту и долготу), вычислить высоту точки земной поверхности над уровнем моря, создать различные визуализации местоположения и получить информацию о часовом поясе для произвольного расположения и другие виды информации.

Пытаясь перенести эти примеры в собственную практику, перед отправкой запроса не забудьте активировать необходимые API на консоли Google API. Google использует активацию API для вычисления собственных метрик («Сколько пользователей воспользовались этим API?»). Поэтому вы должны активировать API перед использованием.

С помощью Google Geocode API вы можете отправить с Вашего браузера простой GET-запрос, чтобы определить координаты любого адреса (в данном случае координаты Бостонского музея науки):

```
https://maps.googleapis.com/maps/api/geocode/json?address=1+Science+Park+Boston
+MA+02114&key=<ваш API-ключ>
```

```
"results" : [ { "address_components" : [ { "long_name" : "Museum Of Science Drive
way", "short_name" : "Museum Of Science Driveway", "types" : [ "route" ] }, { "l
ong_name" : "Boston", "short_name" : "Boston", "types" : [ "locality", "politica
l" ] }, { "long_name" : "Massachusetts", "short_name" : "MA", "types" : [ "admin
istrative_area_level_1", "political" ] }, { "long_name" : "United States", "shor
t_name" : "US", "types" : [ "country", "political" ] }, { "long_name" : "0211
4", "short_name" : "02114", "types" : [ "postal_code" ] } ], "formatted_address"
: "Museum Of Science Driveway, Boston, MA 02114, USA", "geometry" : { "bounds" :
{ "northeast" : { "lat" : 42.368454, "lng" : -71.06961339999999 }, "southwest" :
{ "lat" : 42.3672568, "lng" : -71.0719624 } }, "location" : { "lat" : 42.3677994
, "lng" : -71.0708078 }, "location_type" : "GEOMETRIC_CENTER", "viewport" : { "n
ortheast" : { "lat" : 42.3692043802915, "lng" : -71.06943891970849 }, "southwest
" : { "lat" : 42.3665064197085, "lng" : -71.0721368802915 } } }, "types" : [ "ro
ute" ] } ], "status" : "OK" }
```

Обратите внимание, что адрес, который мы отправили API, задан в свободной форме. Google есть Google, Geocode API великолепно обрабатывает адреса, у которых отсутствуют почтовые индексы или информация о штате (или даже неправильно написанные адреса), и возвращает наилучшее предположение о том, что является правильным адресом. Например, неправильно написанный запрос `1+Skience+Park+Bostton+MA` (даже без почтового индекса) возвращает тот же самый результат.

Я использовала Geocode API несколько раз, не только для того, чтобы отформатировать введенный пользователем адрес на веб-

сайте, но и для поиска адресов в тексте веб-страниц, и применяла API, чтобы преобразовать найденные адреса в формат, наиболее удобный для хранения и поиска.

Чтобы получить информацию о часовом поясе для наших найденных координат, вы можете использовать Time Zone API:

```
https://maps.googleapis.com/maps/api/timezone/json?location=42.3677994,-
71.0708078&timestamp=1412649030&key=<ваш API-ключ>
```

который возвращает ответ:

```
{ "dstOffset" : 3600, "rawOffset" : -18000, "status" : "OK", "timeZon
eId" : "America/New_York", "timeZoneName" : "Eastern Daylight Time" }
```

Для отправки запроса к Time Zone API требуется временная метка UNIX. Это позволяет Google предоставить вам информацию о часовом поясе с учетом перехода на летнее время. Даже для тех городов, где часовой пояс не зависит от времени года (например, Феникс, который не использует летнее время), в запросе к API нужно указывать временную метку.

Чтобы завершить этот довольно краткий тур по коллекции инструментов Google Maps API, вы можете вычислить высоту над уровнем моря для данных координат:

```
https://maps.googleapis.com/maps/api/elevation/json?locations=42.3677994,
-71.0708078&key=<ваш API-ключ>
```

В ответ получаем высоту над уровнем моря в метрах, а также разрешение, указывающее расстояние в метрах до самой дальней точки, по которой эта высота интерполируется. Меньшее значение разрешения указывает на более высокую точность измерения данной высоты:

```
{ "results" : [ { "elevation" : 5.127755641937256, "location" : { "la
t" : 42.3677994, "lng" : -71.0708078 }, "resolution" : 9.543951988220
215 } ], "status" : "OK" }
```

Парсинг JSON-данных

В этой главе мы рассмотрели различные типы API и их работу и привели примеры ответов API в формате JSON. Теперь давайте посмотрим, как мы можем распарсить и использовать эту информацию.

В начале главы я приводила пример с *freegeoip.net*, который преобразует IP-адреса в физические адреса:

```
http://freegeoip.net/json/50.78.253.58
```

Я могу взять вывод этого запроса и применить для его расшифровки питоновские функции, выполняющие парсинг JSON-данных:

```
import json
from urllib.request import urlopen

def getCountry(ipAddress):
    response = urlopen("http://freegeoip.net/json/"+ipAddress).read()
                       .decode('utf-8')
    responseJson = json.loads(response)
    return responseJson.get("country_code")

print(getCountry("50.78.253.58"))
```

Этот код выводит код страны для IP-адреса: *50.78.253.58*.

Библиотека, используемая для парсинга JSON-данных, является частью стандартной библиотеки языка Python. Просто сначала введите `import json`, и все готово! В отличие от многих языков, которые могут распарсить JSON-данные в специальный JSON-объект, Python использует более гибкий подход и превращает JSON-объекты в словари, JSON-массивы – в списки, JSON-строки – в строки и т. д. Таким образом, мы можем легко получить доступ к значениям, хранящимся в формате JSON, и совершать различные операции с ними.

Код, приведенный ниже, дает краткое представление о том, как JSON-библиотека языка Python обрабатывает различные значения в JSON-строке:

```
import json

jsonString = '{"arrayOfNums":[{"number":0},{"number":1},{"number":2}],
              "arrayOfFruits":{"fruit":"apple"},{"fruit":"banana"},
                              {"fruit":"pear"}]}'
jsonObj = json.loads(jsonString)

print(jsonObj.get("arrayOfNums"))
print(jsonObj.get("arrayOfNums")[1])
print(jsonObj.get("arrayOfNums")[1].get("number")+
      jsonObj.get("arrayOfNums")[2].get("number"))
print(jsonObj.get("arrayOfFruits")[2].get("fruit"))
```

Вывод выглядит следующим образом:

```
[{'number': 0}, {'number': 1}, {'number': 2}]
{'number': 1}
3
pear
```

Строка 1 представляет собой список объектов словаря, строка 2 – объект словаря, строка 3 – целое число (сумма целых чисел в словарях), и строка 4 – это строка.

Возвращаем все это домой

Несмотря на то что смысл существования многих современных веб-приложений заключается в том, чтобы взять имеющиеся данные и преобразовать их в более привлекательный формат, я бы сказала, что в большинстве случаев это не самое интересное занятие. Если вы используете API в качестве единственного источника данных, самое оптимальное – это просто скопировать уже имеющуюся и, по сути, опубликованную базу данных. Гораздо интереснее взять два или более источника данных и объединить их или использовать API в качестве инструмента, чтобы посмотреть на собранные данные с нового ракурса.

Давайте рассмотрим пример того, как данные, полученные с помощью API, можно использовать в сочетании с веб-скрапингом: например, чтобы увидеть, пользователи каких стран вносят наибольший вклад в составление Википедии.

Анализируя Википедию, вы, вероятно, уже видели страницу истории правок статьи, которая отображает список последних изменений. Если пользователи, желая внести изменения в статью, заходят в Википедию под своим логином, в списке правок будут зафиксированы их имена. Если вы вносите изменения, зайдя на сайт Википедии, не логинясь, записывается IP-адрес, как показано на рис. 4.4.

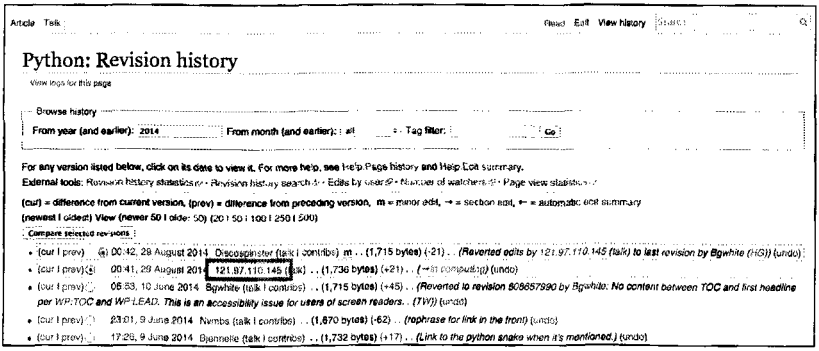


Рис. 4.4 ❖ IP-адрес анонимного автора правки, показанный на странице истории изменений для статьи Python

IP-адрес, обведенный рамкой на странице истории, – *121.97.110.145*. С помощью *freegeoip.net* API выясняем, что данный IP-адрес на момент написания книги (географическое местоположение IP-адреса может иногда меняться) был привязан к городу Кесон-Сити (Филиппины).

Сама по себе эта информация не столь интересна, но что, если мы соберем информацию о географическом местоположении по большому количеству авторов правок? Несколько лет назад я довольно просто сделала это, используя библиотеку Google Geochart (http://bit.ly/1ADw9Dd), чтобы построить любопытный график (http://bit.ly/1cs2CAK), который показывает географическое местоположение авторов правок с учетом языка редактирования (рис. 4.5).

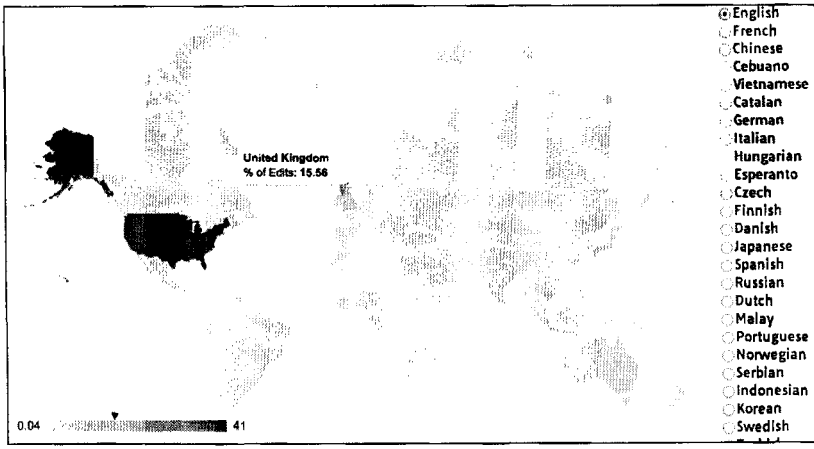


Рис. 4.5 ❖ Визуализация географического местоположения авторов правок Википедии с использованием библиотеки Geochart компании Google

Создание основного скрипта, который перемещается по страницам Википедии, ищет страницы истории правок, а затем находит IP-адреса авторов правок на этих страницах, не является сложной задачей. Используя модифицированный код из главы 3, скрипт, приведенный ниже, выполняет следующие операции:

```
from urllib.request import urlopen
from bs4 import BeautifulSoup
import datetime
import random
```

```
import re

random.seed(datetime.datetime.now())
def getLinks(articleUrl):
    html = urlopen("http://en.wikipedia.org"+articleUrl)
    bsObj = BeautifulSoup(html)
    return bsObj.find("div", {"id":"bodyContent"}).findAll("a",
                    href=re.compile("^(/wiki/)((?!:).)*$"))

def getHistoryIPs(pageUrl):
    #Формат страниц истории выглядит следующим образом:
    #http://en.wikipedia.org/w/index.php?title=Title_in_URL&action=history
    pageUrl = pageUrl.replace("/wiki/", "")
    historyUrl = "http://en.wikipedia.org/w/index.php?title="
                 +pageUrl+"&action=history"
    print("history url is: "+historyUrl)
    html = urlopen(historyUrl)
    bsObj = BeautifulSoup(html)
    #находит лишь ссылки с классом "mw-anonuserlink", которые
    #содержат IP-адреса вместо имен пользователей
    ipAddresses = bsObj.findAll("a", {"class":"mw-anonuserlink"})
    addressList = set()
    for ipAddress in ipAddresses:
        addressList.add(ipAddress.get_text())
    return addressList

links = getLinks("/wiki/Python_(programming_language)")

while(len(links) > 0):
    for link in links:
        print("-------------------")
        historyIPs = getHistoryIPs(link.attrs["href"])
        for historyIP in historyIPs:
            print(historyIP)

    newLink = links[random.randint(0, len(links)-1)].attrs["href"]
    links = getLinks(newLink)
```

Эта программа использует две основные функции: функцию `getLinks` (которая также использовалась в главе 3) и новую функцию `getHistoryIPs`, которая ищет содержимое всех ссылок с классом `mw-anonuserlink` (указывает анонимного пользователя с IP-адресом, а не имя пользователя) и возвращает его в виде набора.

Поговорим о наборах

До этого момента я использовала исключительно две структуры данных Python: списки и словари. При наличии этих двух структур зачем еще использовать набор?

Наборы Python не упорядочены, то есть вам не нужно ссылаться на конкретную позицию в наборе, чтобы получить искомое значение. Порядок, в котором вы добавляете элементы в набор, не обязательно должен соответствовать порядку, в котором вы их возвращаете. Одним из замечательных свойств набора, которым я воспользовалась в примере кода, является то, что в него не надо многократно включать один и тот же элемент. Если вы добавите в набор строку, которая в нем уже есть, она не будет дублироваться. Таким образом, я могу быстро извлечь из страницы истории правок список лишь уникальных IP-адресов, не принимая во внимание многочисленные правки, внесенные одним и тем же пользователем.

Выбирая между наборами и списками в коде, который необходимо масштабировать, помните о нескольких вещах: несмотря на то что списки перебираются немного быстрее, наборы чуть удобнее в плане поиска (определяющего наличие или отсутствие объекта в наборе).

Кроме того, этот код использует несколько произвольный (но эффективный для данного примера) шаблон поиска статей, из которых нужно извлечь истории правок. Он начинает свою работу с извлечения историй правок из всех статей Википедии, связанных со стартовой страницей (в данном случае речь идет о статье, посвященной языку программирования Python). Потом он случайно выбирает новую стартовую страницу и извлекает все истории правок из статей, связанных с этой страницей. Так продолжается до тех пор, пока он не попадет на страницу без ссылок.

Теперь, когда у нас есть код, который извлекает IP-адреса в виде строки, мы можем скомбинировать его с функцией `getCountry` из предыдущего раздела, чтобы определить по этим IP-адресам страны. Я слегка модифицировала `getCountry`, чтобы учесть недействительные или неправильные IP-адреса, которые приведут к возникновению ошибки «404 Not Found» (например, на момент написания книги *freegeoip.net* не работал с IPv6, что могло повлечь возникновение данной ошибки):

```
def getCountry(ipAddress):
    try:
        response = urlopen("http://freegeoip.net/json/"
                           +ipAddress).read().decode('utf-8')
    except HTTPError:
        return None
    responseJson = json.loads(response)
    return responseJson.get("country_code")

links = getLinks("/wiki/Python_(programming_language)")

while(len(links) > 0):
    for link in links:
        print("----------------")
```

```
        historyIPs = getHistoryIPs(link.attrs["href"])
        for historyIP in historyIPs:
            country = getCountry(historyIP)
            if country is not None:
                print(historyIP+" is from "+country)

    newLink = links[random.randint(0, len(links)-1)].attrs["href"]
    links = getLinks(newLink)
```

Полный исполняемый код можно посмотреть на странице http://www.pythonscraping.com/code/6-3.txt.

Ниже приводится пример вывода:

```
------------------
history url is: http://en.wikipedia.org/w/index.php?title=Programming_
paradigm&action=history
68.183.108.13 is from US
86.155.0.186 is from GB
188.55.200.254 is from SA
108.221.18.208 is from US
141.117.232.168 is from CA
76.105.209.39 is from US
182.184.123.106 is from PK
212.219.47.52 is from GB
72.27.184.57 is from JM
49.147.183.43 is from PH
209.197.41.132 is from US
174.66.150.151 is from US
```

Подробнее о применении API

В этой главе мы рассмотрели несколько современных интерфейсов, которые используются для получения доступа к данным в Интернете, в частности поработали с API, которые могут быть полезны для веб-скрапинга. Однако я боюсь, что приведенные примеры не раскрывают в полной мере широкого определения «обмен данными между разнообразным программным обеспечением».

Поскольку эта книга посвящена веб-скрапингу и она не призвана служить общим руководством по сбору данных, я могу лишь вам посоветовать несколько отличных ресурсов для дальнейшего изучения данной темы, если это потребуется.

В книге Леонарда Ричардсона, Майка Амундсена и Сэма Руби *RESTful Web APIs* дается подробный обзор теоретических и практических аспектов использования API в Интернете. Кроме того, у Май-

ка Амундсена есть увлекательные видеокурсы *Designing APIs for the Web*, которые научат вас создавать свои собственные интерфейсы, полезная вещь, если вы решили опубликовать ваши собранные данные в удобном формате.

На первый взгляд, веб-скрапинг и Web API могут показаться совершенно разными темами. Однако я надеюсь, что эта глава показала, что они дополняют друг друга, образуя непрерывный процесс сбора данных. В каком-то смысле Web API можно расматривать как поднаправление веб-скрапинга. Ведь в конечном счете вы пишете скрипт, который собирает данные с удаленного веб-сервера и анализирует их в удобном формате, как если бы мы это сделали с помощью веб-скрапера.

Глава 5

Хранение данных

Несмотря на то что вывод результатов в терминале довольно забавен, это невероятно неудобно, когда дело касается агрегации и анализа данных. Чтобы веб-скраперы были мало-мальски полезны, вы должны уметь сохранить собранную ими информацию.

В этой главе мы рассмотрим три основных метода управления данными, которых будет достаточно для практически любого воображаемого приложения. Вы хотите управлять внутренней (административной) частью сайта или создать свой собственный API? Вам, вероятно, нужны скраперы, которые могут записать собранную информацию в базу данных. Нужен быстрый и легкий способ скачать некоторые документы из Интернета и записать их на ваш жесткий диск? Вы, вероятно, захотите создать файл-поток. Нужны уведомления о событиях, данные агрегируются один раз в день? Отправьте себе электронное письмо!

Помимо веб-скрапинга, возможность хранить и работать с большими объемами данных невероятно важна для любого современного программного приложения. Фактически информация этой главы необходима для реализации различных примеров, которые будут даваться в последующих разделах книги. Я настоятельно рекомендую, по крайней мере, пробежать глазами эту главу, если вы незнакомы с автоматизированным хранением данных.

Медиафайлы

Есть два основных способа хранить мультимедийные файлы: в виде ссылки или загрузив сам файл. Вы можете хранить файл в виде ссылки, просто сохранив URL-адрес файла. Это дает несколько преимуществ:

- скраперы будут работать намного быстрее, ведь когда им не нужно загружать файлы, требуется гораздо меньшая пропускная способность;

- вы экономите место на вашем компьютере, сохраняя только URL-адреса;
- код, который только хранит URL-адреса и не должен загружать файлы, проще написать;
- вы можете уменьшить нагрузку на хост-сервер, избежав загрузки больших файлов.

Ниже приводятся недостатки:

- включение URL-адресов сторонних ресурсов в ваш собственный веб-сайт или приложение известно как *хотлинкинг* (hotlinking), и это очень быстрый способ влипнуть в неприятную историю, связанную с Интернетом;
- вам не следует использовать сторонние серверы для размещения медиаконтента ваших собственных приложений;
- файл, размещенный по конкретному URL-адресу, может измениться. Это может привести к недоразумениям, если, скажем, вы разместите изображение с чужого сервера в публичном блоге. Если вы сохраняете URL-адреса, с тем чтобы скачать файл позже, для дальнейшего анализа, в конечном итоге файл может быть удален или изменен, что сделает его анализ на более позднем этапе нерелевантным;
- в реальности веб-браузеры не просто запрашивают HTML-страницу и переходят на нее – они загружают все элементы страницы. Скачивание файлов позволит сделать работу скрапера похожей на работу человека, просматривающего сайт, и это может стать преимуществом.

Задумываясь над тем, как хранить данные – в виде файла или просто URL-адреса файла, вы должны спросить себя, обращусь я к этим файлам больше одного или двух раз или же эта база данных большую часть времени будет сборищем электронного мусора. Если вы выбрали последнее, то, наверное, лучше просто хранить URL-адреса. Если вы выбрали первое, читайте дальше!

В Python 3.x для скачивания файлов с любого удаленного URL-адреса можно использовать `urllib.request.urlretrieve`:

```
from urllib.request import urlretrieve
from urllib.request import urlopen
from bs4 import BeautifulSoup

html = urlopen("http://www.pythonscraping.com")
bsObj = BeautifulSoup(html)
imageLocation = bsObj.find("a", {"id": "logo"}).find("img")["src"]
urlretrieve (imageLocation, "logo.jpg")
```

Этот код скачивает логотип из *http://pythonscraping.com* и сохраняет его в виде файла *logo.jpg* в том же самом каталоге, из которого запущен скрипт.

Он хорошо работает, если вам нужно скачать только один файл и вы знаете его имя и расширение. Но большинство скраперов не скачивают один файл и затем завершают свою работу. Код, приведенный ниже, скачивает с домашней страницы *http://pythonscraping.com* все внутренние файлы в тегах с атрибутом `src`:

```
import os
from urllib.request import urlretrieve
from urllib.request import urlopen
from bs4 import BeautifulSoup

downloadDirectory = "downloaded"
baseUrl = "http://pythonscraping.com"

def getAbsoluteURL(baseUrl, source):
    if source.startswith("http://www."):
        url = "http://"+source[11:]
    elif source.startswith("http://"):
        url = source
    elif source.startswith("www."):
        url = source[4:]
        url = "http://"+source
    else:
        url = baseUrl+"/"+source
    if baseUrl not in url:
        return None
    return url

def getDownloadPath(baseUrl, absoluteUrl, downloadDirectory):
    path = absoluteUrl.replace("www.", "")
    path = path.replace(baseUrl, "")
    path = downloadDirectory+path
    directory = os.path.dirname(path)

    if not os.path.exists(directory):
        os.makedirs(directory)

    return path

html = urlopen("http://www.pythonscraping.com")
bsObj = BeautifulSoup(html)
downloadList = bsObj.findAll(src=True)

for download in downloadList:
    fileUrl = getAbsoluteURL(baseUrl, download["src"])
    if fileUrl is not None:
        print(fileUrl)

urlretrieve(fileUrl, getDownloadPath(baseUrl, fileUrl, downloadDirectory))
```

Запускайте код с осторожностью

Вам знакомы все эти предостережения, связанные со скачиванием неизвестных файлов из Интернета? Вышеприведенный скрипт скачает на жесткий диск Вашего компьютера все, что попадется на его пути. Включая случайные Bash-скрипты, exe-файлы и другие потенциально вредоносные программы.

Думаете, что вы в безопасности, потому что вы на самом деле никогда бы и не запустили тот или иной файл, отправленный в папку загрузок? Особенно вы рискуете, запуская вышеприведенную программу от имени администратора. Что произойдет, если при сканировании веб-сайта вы наткнетесь на файл, который отправляет сам себя по адресу *../../../../usr/bin/python*? При следующем запуске скрипта Python из командной строки вы фактически запустите вредоносную программу на вашем компьютере! Эта программа приведена только для иллюстрации. Ее нельзя применять наобум, без тщательной проверки имен файлов, и ее нужно запускать от имени учетной записи с ограниченными правами. Как обычно, резервное копирование файлов, отсутствие конфиденциальной информации на вашем жестком диске и немного здравого смысла значат очень многое.

Этот скрипт использует лямда-функцию (о лямбда-функциях рассказывалось в главе 2), чтобы выбрать все теги на главной странице, которые имеют атрибут `src`, затем очищает и нормализует URL-адреса, чтобы получить абсолютный путь для каждой закачки (за исключением внешних ссылок). Затем каждый файл скачивается в локальную папку *downloaded* на вашем компьютере.

Обратите внимание, что питоновский модуль os помещает каждый файл в соответствующий каталог и попутно создает недостающие каталоги, если это необходимо. Модуль os выступает в качестве интерфейса между Python и операционной системой, что позволяет управлять путями к файлам, создавать каталоги, получать информацию о запущенных процессах и переменных окружения и выполнять множество других полезных вещей.

Сохранение данных в формате CSV

CSV (Comma Separated Values – значения, разделенные запятыми) является одним из самых популярных файловых форматов, в котором хранятся данные электронных таблиц. Он поддерживается Microsoft Excel и многими другими приложениями из-за своей простоты. Ниже приведен пример обычного файла CSV:

```
fruit,cost
apple,1.00
banana,0.30
pear,1.25
```

Как и в Python, пробельный символ (whitespace) здесь очень важен. Каждая строка отделена символом новой строки, в то время как столбцы в строке разделены запятыми (отсюда название «значения, разделенные запятыми»). Другие виды CSV-файлов (иногда их называют файлами со «значениями, разделенными символами») используют табуляцию или другие символы в качестве разделителей, однако эти форматы менее распространены и не так широко поддерживаются.

Если вы хотите скачать CSV-файлы напрямую из Интернета и хранить их локально, не прибегая к парсингу или изменению, вам этот раздел не понадобится. Просто скачайте их, как любой другой файл, и сохраните их в формате CSV-файла с помощью методов, описанных в предыдущем разделе.

Изменить CSV-файл или даже создать его с нуля необычайно просто, воспользовавшись питоновской библиотекой csv:

```
import csv

csvFile = open("../files/test.csv", 'wt')
try:
    writer = csv.writer(csvFile)
    writer.writerow(('number', 'number plus 2', 'number times 2'))
    for i in range(10):
        writer.writerow( (i, i+2, i*2))
finally:
    csvFile.close()
```

Предупреждаем: создание файла в Python является обязательным условием. Если *../files/test.csv* не существует, Python создаст каталог и файл автоматически. Если он уже существует, Python перезапишет *test.csv* новыми данными.

После выполнения кода вы должны посмотреть CSV-файл:

```
number,number plus 2,number times 2
0,2,0
1,3,2
2,4,4
...
```

Одна из распространенных задач веб-скрапинга заключается в том, чтобы извлечь HTML-таблицу и записать ее в виде CSV-файла. Страница Википедии Comparison of Text Editors содержит довольно сложную HTML-таблицу в сочетании с цветовой кодировкой, ссылками, сортировкой и другими ненужными элементами HTML, от которых надо избавиться, прежде чем эта таблица будет записана в фор-

мате CSV. Активно используя библиотеку BeautifulSoup и функцию get_text(), вы можете выполнить эту задачу, обойдясь менее чем 20 строками:

```
import csv
from urllib.request import urlopen
from bs4 import BeautifulSoup

html = urlopen("http://en.wikipedia.org/wiki/Comparison_of_text_editors")
bsObj = BeautifulSoup(html)
#В данный момент главная таблица сравнений является первой таблицей
#на странице
table = bsObj.findAll("table",{"class":"wikitable"})[0]
rows = table.findAll("tr")

csvFile = open("../files/editors.csv", 'wt')
writer = csv.writer(csvFile)
try:
    for row in rows:
    csvRow = []
    for cell in row.findAll(['td', 'th']):
        csvRow.append(cell.get_text())
        writer.writerow(csvRow)
finally:
    csvFile.close()
```

Перед тем, как применить этот код в реальной жизни

Этот скрипт можно прекрасно использовать в скраперах при работе с большим количеством HTML-таблиц, которые нужно преобразовать в CSV-файлы, или большим количеством HTML-таблиц, которые нужно собрать в один CSV-файл. Однако если вы работаете с одной таблицей, лучший инструмент – это копирование и вставка. Выделение и копирование всего содержимого HTML-таблицы с последующей вставкой в Excel поможет вам получить нужный CSV-файл без запуска скрипта!

Результатом должен быть хорошо отформатированный CSV-файл, сохраненный локально по адресу *../files/editors.csv*, что идеально для отправки и обмена с людьми, которые еще не вполне уверенно владеют MySQL!

MySQL

На сегодняшний день MySQL (официально произносится как «Май-Эс-Кью-Эль», хотя многие говорят «Май-Сикл») является самой популярной системой управления базами данных (СУБД) с открытым исходным кодом. Исторически популярность MySQL росла наравне

с двумя другими крупными системами баз данных с закрытым исходным кодом – Microsoft SQL Server и Oracle, что, принимая во внимание наличие таких крупных конкурентов, несколько необычно для проекта с открытым исходным кодом.

Популярность MySQL имеет свои причины. В большинстве приложений при работе с MySQL трудно допустить ошибки. Это масштабируемая, надежная и полнофункциональная СУБД, используемая ведущими сайтами, например YouTube[1], Twitter[2] и Facebook[3], наряду со многими другими.

Из-за своей вездесущности, цены («свободное распространение» – это довольно привлекательная цена) и удобства использования MySQL создает замечательную базу данных для хранения проектов по веб-скрапингу, и в оставшейся части этой книги мы будем использовать ее.

«Реляционная» база данных?

«Реляционные данные» – это данные, которые содержат взаимосвязи. Рада, что мы прояснили это!

Шучу! Когда программисты заводят разговор о реляционных данных, они обращаются к данным, которые не могут существовать в вакууме. Интересующие нас данные имеют свойства, связывающие их с другими частями данных. Например, «Пользователь А идет учиться в учреждение В», где пользователя А можно найти в таблице «пользователей» базы данных, а учреждение В можно найти в таблице «учреждений» базы данных.

Позже в этой главе мы рассмотрим моделирование различных типов взаимосвязей и эффективные способы хранения данных в MySQL (или любой другой реляционной базе данных).

Установка MySQL

Если вы – новичок в MySQL, установка базы данных может немного напугать вас (если вы хорошо знакомы с MySQL, смело пропускайте этот раздел). На самом деле это так же просто, как установка любого другого программного обеспечения. По сути, MySQL представляет собой набор файлов данных, которые размещены на сервере или локальном компьютере и содержат всю информацию, хранящуюся в вашей базе данных. Кроме того, программный уровень MySQL

---

[1] *Jackson J.* YouTube Scales MySQL with Go Code // PCWorld. December 15. 2012 (http://bit.ly/1LWVmc8).

[2] *Cole J., Arnaut D.* MySQL at Twitter // The Twitter Engineering Blog. April 9. 2012 (http://bit.ly/1KHDKns).

[3] MySQL and Database Engineering: Mark Callaghan // March 4. 2012 (http://on.fb.me/1RFMqvw).

предлагает удобный способ взаимодействия с данными с помощью интерфейса командной строки. Например, команда, приведенная ниже, возвращает список всех пользователей в вашей базе данных с именем «Ryan»:

```
SELECT * FROM users WHERE firstname = "Ryan"
```

В Linux установка MySQL такая же легкая:

```
$sudo apt-get install mysql-server
```

Просто наблюдайте за процессом установки, задайте требования к памяти и введите новый пароль для нового root-пользователя при запросе.

В Mac OS X и Windows есть некоторые сложности. Если у вас еще не установлен MySQL, вам нужно создать учетную запись Oracle перед загрузкой пакета.

Если вы работаете в Mac OS X, вам нужно сначала скачать установочный пакет.

Выберите пакет *.dmg* и войдите в ваш аккаунт Oracle или создайте его, чтобы загрузить файл. Запустив файл, вы должны выполнить шаги довольно простого Мастера установки (рис. 5.1):

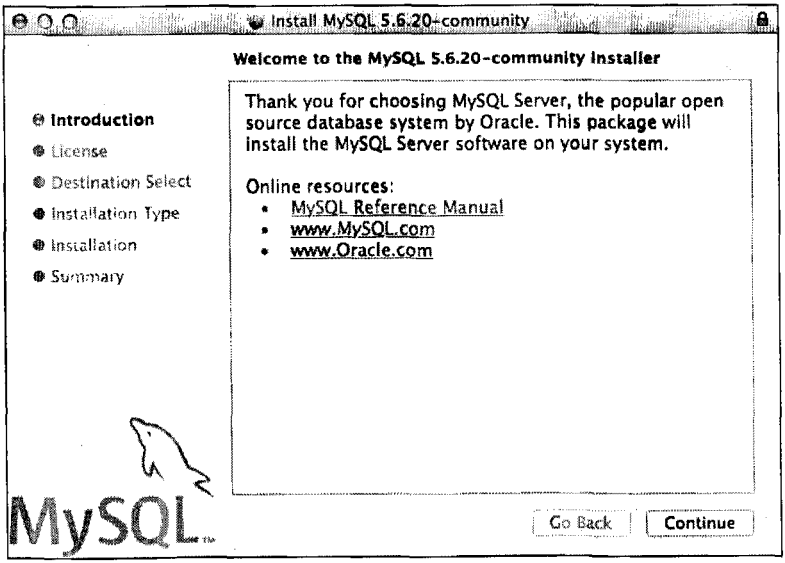


Рис. 5.1 ❖ Установщик MySQL для Mac OS X

Описания шагов обычной установки будет достаточно, и я в рамках этой книги буду исходить из того, что вы выполняете установку MySQL по умолчанию.

Если скачивание и запуск установщика кажутся немного утомительными, вы всегда можете установить менеджер пакетов Homebrew (http://brew.sh/). Установив Homebrew, вы также можете инсталлировать MySQL, запустив:

```
$brew install mysql
```

Homebrew – это отличный проект с открытым кодом и прекрасной поддержкой пакетов Python. Фактически большинство модулей Python, используемых в этой книге, можно очень легко установить с помощью Homebrew. Если у вас он еще не установлен, я настоятельно рекомендую его!

Установив MySQL в Mac OS X, вы можете запустить сервер MySQL следующим образом:

```
$cd /usr/local/mysql
$sudo ./bin/mysqld_safe
```

В Windows установка и запуск MySQL выглядят немного сложнее, однако хорошей новостью является то, что есть удобный установщик (http://bit.ly/1FVKDP9), который упрощает процесс. После завершения загрузки он предложит вам пройти несколько шагов (см. рис. 5.2).

Вы должны установить MySQL, используя настройки по умолчанию, за одним исключением: на странице **Setup Type** (Тип установки) я рекомендую вам выбрать **Server Only** (Только сервер), чтобы избежать установки дополнительного программного обеспечения и библиотек Microsoft.

Далее вы должны использовать настройки установки по умолчанию и следовать инструкциям, чтобы начать работу с сервером MySQL.

Некоторые основные команды

После запуска сервера MySQL у вас появляется множество вариантов работы с базой данных. Существует довольно много программных средств, которые выступают в качестве посредника, поэтому вам необязательно использовать команды MySQL (или, по крайней мере, применять их реже). Такие инструменты, как phpMyAdmin и MySQL Workbench, могут упростить просмотр, сортировку и вставку данных. Однако уметь работать с командной строкой все же важно.

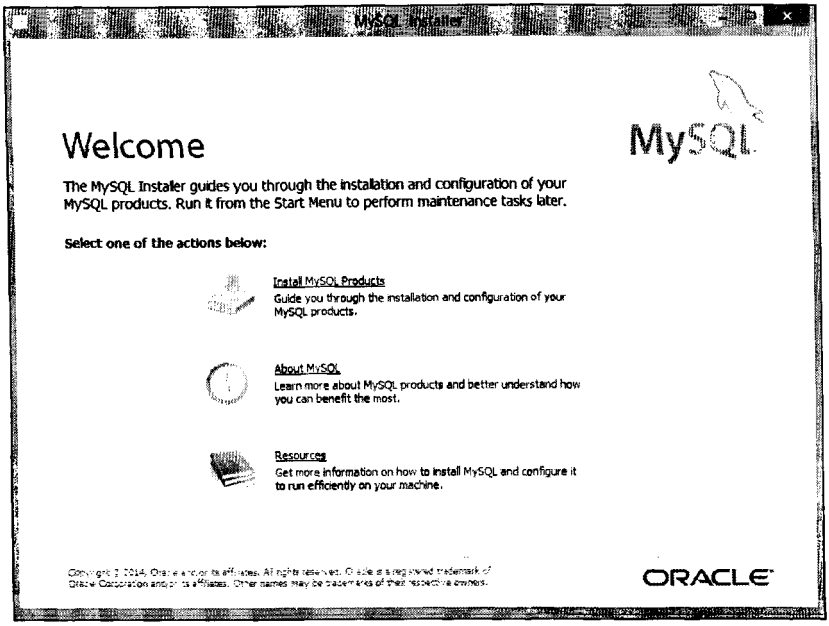


Рис. 5.2 ❖ Установщик MySQL для Windows

За исключением имен переменных, MySQL нечувствителен к регистру. Например, `SELECT` – это то же самое, что и `sElEcT`. Однако когда вы пишете оператор MySQL, все ключевые слова MySQL должны быть в верхнем регистре. При работе с таблицами и базами данных большинство разработчиков предпочитает задавать имена в нижнем регистре, хотя этот стандарт часто игнорируется.

При первом запуске в MySQL базы данных отсутствуют. Вы можете создать базу данных следующим образом:

```
>CREATE DATABASE scraping;
```

Поскольку каждый экземпляр MySQL может иметь несколько баз данных, прежде чем начать работать с базой данных, нужно указать, какие базы данных MySQL мы хотим использовать:

```
>USE scraping;
```

С этого момента (по крайней мере, пока мы не закроем соединение с MySQL или не переключимся на другую базу данных) все вводимые команды применяются к новой базе данных scraping.

Это все кажется довольно простым. Должно быть так же легко создать таблицу в базе данных, не правда ли ли? Давайте попробуем создать таблицу, в которой сохраним коллекцию собранных веб-страниц:

```
>CREATE TABLE pages;
```

Этот код приводит к ошибке:

```
ERROR 1113 (42000): A table must have at least 1 column
```

В отличие от базы данных, которая может не иметь таблиц, таблица в MySQL должна содержать столбцы. Чтобы задать столбцы в MySQL, необходимо указать их через запятую в списке, помещенном в круглые скобки, после оператора CREATE TABLE <tablename>:

```
>CREATE TABLE pages (id BIGINT(7) NOT NULL AUTO_INCREMENT, title VARCHAR(200),
content VARCHAR(10000), created TIMESTAMP DEFAULT CURRENT_TIMESTAMP, PRIMARY KEY
(id));
```

Каждое определение столбца состоит из трех частей:

- имя (id, title, created и т. д.);
- тип переменной (BIGINT(7), VARCHAR, TIMESTAMP);
- любые дополнительные атрибуты (NOT NULL AUTO_INCREMENT) по желанию.

В конце списка вы должны задать «ключ» таблицы. MySQL использует ключи, чтобы упорядочить содержимое таблицы для быстрого поиска. Позже в этой главе я опишу, как использовать эти ключи с пользой для ускорения работы с базами данных, но сейчас оптимальный способ – это использовать в качестве ключа столбец id.

После выполнения запроса вы в любой момент можете посмотреть, как выглядит структура таблицы, используя DESCRIBE:

```
> DESCRIBE pages;
+---------+---------------+------+-----+-------------------+----------------+
| Field   | Type          | Null | Key | Default           | Extra          |
+---------+---------------+------+-----+-------------------+----------------+
| id      | bigint(7)     | NO   | PRI | NULL              | auto_increment |
| title   | varchar(200)  | YES  |     | NULL              |                |
| content | varchar(10000)| YES  |     | NULL              |                |
| created | timestamp     | NO   |     | CURRENT_TIMESTAMP |                |
+---------+---------------+------+-----+-------------------+----------------+
4 rows in set (0.01 sec)
```

Конечно, это по-прежнему пустая таблица. Вы можете вставить тестовые данные в таблицу страниц[1] с помощью следующей строки:

[1] Таблица, которая содержит страницы сайта, собранные в ходе скрапинга.

```
> INSERT INTO pages (title, content) VALUES ("Test page title", "This is some te
st page content. It can be up to 10,000 characters long.");
```

Обратите внимание, несмотря на то что таблица имеет четыре столбца (id, title, content, created), вы должны задать только два из них (title и content), чтобы вставить строку. Это обусловлено тем, что столбец id является автоинкрементным (при добавлении новой строки MySQL автоматически увеличивает значение этого столбца на 1) и в целом он сам может позаботиться о себе. Кроме того, для столбца timestamp значение по умолчанию представляет собой текущее значение даты и времени.

Конечно, мы можем переопределить эти значения по умолчанию:

```
INSERT INTO pages (id, title, content, created) VALUES (3, "Test page title", "
This is some test page content. It can be up to 10,000 characters long.", "2014-
09-21 10:25:32");
```

Этот запрос выполнится, только если в таблице ещё нет строки с указанным значением столбца id. Однако так делать считается плохой практикой; лучше позволить MySQL самой обработать столбцы id и timestamp, если нет веских причин сделать это иначе.

Теперь, когда наша таблица содержит некоторые данные, можно использовать широкий спектр методов отбора этих данных. Ниже приводится несколько примеров использования операторов SELECT:

```
>SELECT * FROM pages WHERE id = 2;
```

Этот оператор сообщает MySQL «Выберите в таблице все строки, в столбце id которых стоит 2». Звездочка (*) действует в качестве группового символа, возвращая все строки, где условие истинно (WHERE id=2). Она возвращает вторую строку в таблице или пустой результат, если нет ни одной строки с id, равным 2. Например, следующий нечувствительный к регистру запрос возвращает все строки, в столбце title которых содержится текст «test» (знак % выступает в качестве группового символа в MySQL):

```
>SELECT * FROM pages WHERE title LIKE "%test%";
```

Ну а если у вас есть таблица с большим количеством столбцов и вам нужна только определенная часть данных? Вы можете сделать следующее:

```
>SELECT id, title FROM pages WHERE content LIKE "%page content%";
```

Этот код возвращает только столбцы id и title, где есть фраза «page content». Операторы DELETE используют почти такой же синтаксис, что и операторы SELECT:

```
>DELETE FROM pages WHERE id = 1;
```

По этой причине рекомендуется (особенно при работе с важными базами данных, которые нельзя легко восстановить) написать сначала операторы SELECT (в данном случае SELECT * FROM pages WHERE id = 1), протестировать их, убедившись, что возвращаются только те строки, которые вы хотите удалить, а затем заменить SELECT * на DELETE. У многих программистов есть свои страшки, связанные с тем, что они задали неверное условие в операторе DELETE или, что еще хуже, вообще не задали условия, когда спешили, и в итоге уничтожили данные о клиентах. Не делайте так!

Аналогичные меры предосторожности должны быть приняты в отношении операторов UPDATE:

```
>UPDATE pages SET title="A new title", content="Some new content" WHERE id=2;
```

В рамках этой книги мы будем работать только с простыми операторами MySQL, выполняющими отбор, вставку и изменение. Если вы хотите более подробно изучить этот мощный инструмент управления базами данных, я рекомендую книгу Пола Дюбуа *MySQL Cookbook*.

Интеграция с Python

К сожалению, поддержка Python в MySQL не предусмотрена. Однако существует большое количество библиотек с открытым исходным кодом как для Python 2.x, так и для Python 3.x, которые можно использовать для работы с базой данных MySQL. Одной из самых популярных библиотек является PyMySQL.

На момент написания книги была доступна PyMySQL версии 0.6.2, и вы можете скачать и установить эту версию с помощью следующих команд:

```
$ curl -L https://github.com/PyMySQL/PyMySQL/tarball/pymysql-0.6.2 | tar xz
$ cd PyMySQL-PyMySQL-f953785/
$ python setup.py install
```

Не забудьте проверить последнюю версию PyMySQL на сайте и, если необходимо, измените номер версии в первой строке этой команды.

После установки вы автоматически получите доступ к пакету PyMySQL и, запустив локальный сервер MySQL, сможете успешно выполнить следующий скрипт (не забудьте добавить root-пароль для вашей базы данных):

```
import pymysql
conn = pymysql.connect(host='127.0.0.1', unix_socket='/tmp/mysql.sock',
                       user='root', passwd=None, db='mysql')
```

```
cur = conn.cursor()
cur.execute("USE scraping")
cur.execute("SELECT * FROM pages WHERE id=1")
print(cur.fetchone())
cur.close()
conn.close()
```

В этом примере используются два новых типа объектов: объект Connection (`conn`) и объект Cursor (`cur`).

Модель подключение/курсор широко используется в программировании баз данных, хотя некоторым пользователям она может показаться сложной, поскольку сначала нужно понять разницу между подключением и курсором. Подключение отвечает, разумеется, за подключение к базе данных, а также за передачу информации базы данных, выполнение откатов (если запрос или набор запросов нужно прервать и база данных должна быть возвращена в исходное состояние) и создание новых курсоров. Курсор предназначен для работы с базой данных.

Подключение может иметь много курсоров. Курсор отслеживает определенную информацию *состояния*, например какую базу данных он использует. Если у вас есть несколько баз данных и нужно записать информацию во все базы, вам, возможно, понадобится несколько курсоров для выполнения этой задачи. Курсор также содержит результаты последнего выполненного запроса. Вызвав функцию для курсора `cur.fetchone()`, вы можете получить доступ к этой информации.

Важно, чтобы и курсор, и подключение были закрыты после завершения их использования. Если этого не сделать, могут возникнуть *утечки подключений (connection leaks)*, накопление незакрытых соединений, которые больше не используются, но программное обеспечение не в состоянии закрыть их, потому что думает, что вы, возможно, все еще используете их. Это та самая причина, по которой базы данных все время падают, поэтому не забывайте закрывать подключения!

Самое частое, что вы, вероятно, хотите сделать, начав работу с MySQL, – это сохранить результаты скрапинга в базе данных. Давайте посмотрим, как это можно сделать, используя предыдущий пример – скрапер Википедии.

Работа с текстом Unicode, полученным в ходе скрапинга, может быть трудной. По умолчанию MySQL не работает с Unicode. К счастью, вы можете включить эту функцию (только имейте в виду, что это приведет к увеличению размера базы данных). Поскольку мы

должны обработать различные символы Википедии, сейчас самое время познакомить базу данных с текстом Unicode:

```
ALTER DATABASE scraping CHARACTER SET = utf8mb4 COLLATE = utf8mb4_unicode_ci;
ALTER TABLE pages CONVERT TO CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
ALTER TABLE pages CHANGE title title VARCHAR(200) CHARACTER SET utf8mb4 COLLATE
utf8mb4_unicode_ci;
ALTER TABLE pages CHANGE content content VARCHAR(10000) CHARACTER SET utf8mb4 CO
LLATE utf8mb4_unicode_ci;
```

Эти четыре строки меняют кодировку по умолчанию для базы данных, для таблицы и для обоих столбцов с `utf8mb4` (формат хоть и предназначен для Unicode, но у него ужасная поддержка большинства Unicode-символов) на `utf8mb4_unicode_ci`.

Успешность смены кодировки проверяется так: при попытке вставить несколько умлаутов или несколько китайских символов в поле `title` или `content` базы данных не должно возникнуть ошибок.

Теперь, когда база данных готова вобрать в себя все многообразие контента Википедии, вы можете запустить следующий код:

```
from urllib.request import urlopen
from bs4 import BeautifulSoup
import datetime
import random
import pymysql

conn = pymysql.connect(host='127.0.0.1', unix_socket='/tmp/mysql.sock',
                       user='root', passwd=None, db='mysql', charset='utf8')
cur = conn.cursor()
cur.execute("USE scraping")

random.seed(datetime.datetime.now())

def store(title, content):
    cur.execute("INSERT INTO pages (title, content) VALUES (\"%s\",
                \"%s\")", (title, content))
    cur.connection.commit()

def getLinks(articleUrl):
    html = urlopen("http://en.wikipedia.org"+articleUrl)
    bsObj = BeautifulSoup(html)
    title = bsObj.find("h1").find("span").get_text()
    content = bsObj.find("div", {"id":"mw-content-text"}).find("p").get_text()
    store(title, content)
    return bsObj.find("div", {"id":"bodyContent"}).findAll("a",
                      href=re.compile("^(/wiki/)((?!:).)*$"))

links = getLinks("/wiki/Kevin_Bacon")
```

```
try:
    while len(links) > 0:
        newArticle = links[random.randint(0, len(links)-1)].attrs["href"]
        print(newArticle)
        links = getLinks(newArticle)
finally:
    cur.close()
    conn.close()
```

Есть несколько вещей, на которые стоит обратить внимание: во-первых, строка подключения к базе данных содержит `"charset='utf8'"`. Она сообщает, что при подключении всю информация, отправляемая в базу данных, должна быть в формате UTF-8 (и конечно, база данных уже должна быть сконфигурирована для работы с этим форматом).

Во-вторых, обратите внимание на функцию `store`. Она принимает в качестве аргументов два строковые переменные `title` и `content` и добавляет их в оператор `INSERT`, который выполняется с помощью курсора, а затем фиксируется с помощью подключения курсора. Это отличный пример разделения курсора и подключения. Пока курсор содержит некоторую сохраненную информацию о базе данных, он должен работать через подключение, чтобы отправить информацию обратно в базу данных и вставить определенную информацию.

Наконец, вы видите, что оператор `finally` был добавлен в основной цикл программы в нижней части кода. Это гарантирует, что независимо от способа прерывания программы или исключений, которые могут быть сгенерированы в ходе ее исполнения (ну, разумеется, поскольку интернет-данные полны мусора, вы всегда должны быть готовы к выдаче исключений), курсор и соединение будут немедленно закрыты до завершения работы программы. Всякий раз, когда вы выполняете веб-скрапинг и используете открытое подключение к базе данных, рекомендуется использовать оператор `try...finally`.

Хотя PyMySQL не является большим пакетом, он содержит массу полезных функций, описание которых эта книга просто не может вместить. Обратитесь к следующей документации: *http://bit.ly/1KHzoga*.

Методы работы с базами данных и эффективная практика

Существуют специалисты, которые всю свою карьеру тратят на изучение, настройку и проектирование баз данных. Я не отношусь к ним, и эта книга не об этом. Однако, как и во многих областях компьютер-

ных наук, есть несколько приемов, которые можно быстро освоить, чтобы в значительной мере ускорить работу с базами данных в различных приложениях.

Во-первых, всегда добавляйте столбцы `id` в ваши таблицы (за некоторыми исключениями). Все таблицы в MySQL должны иметь, по крайней мере, один первичный ключ (ключевой столбец, по которому MySQL делает сортировку), таким образом, MySQL будет знать, как упорядочить данные по нему. Однако довольно трудно определить, что выбрать в качестве первичного ключа. Дебаты по поводу того, что лучше использовать – искусственно созданный столбец `id` или определенный уникальный атрибут типа `username`, уже бушуют между специалистами по анализу данных и программистами долгие годы, хотя я обычно склоняюсь в пользу того, что нужно создать столбцы `id`. Причины этих споров сложны, но при работе с обычными (некорпоративными) системами баз данных вы всегда должны использовать столбец `id` в качестве автоинкрементного первичного ключа.

Во-вторых, используйте интеллектуальную индексацию. Словарь (в смысле книга, а не объект Python) – это список слов, проиндексированных в алфавитном порядке. Он позволяет выполнить быстрый поиск нужного слова, при условии что вы знаете, как оно пишется. А теперь вообразите себе словарь, который организован в алфавитном порядке по определению слова. Это покажется вам бесполезным, если вы не играли в игру *Jeopardy!*, когда дается определение и нужно подобрать слово. Однако при поиске в БД такие ситуации бывают. Например, Ваша база данных может содержать вот такое поле, которое вы часто будете запрашивать:

```
>SELECT * FROM dictionary WHERE definition="A small furry animal that says meow";
+------+-------+-----------------------------------+
| id   | word  | definition                        |
+------+-------+-----------------------------------+
|  200 | cat   | A small furry animal that says meow |
+------+-------+-----------------------------------+
1 row in set (0.00 sec)
```

Вы, возможно, очень хотели бы добавить дополнительный ключ в эту таблицу (помимо ключа, в качестве которого, по-видимому, задан столбцец `id`), чтобы ускорить поиск по определению столбца. К сожалению, дополнительная индексация требует больше места (из-за создания нового индекса), а также некоторое дополнительное процессорное время при вставке новых строк. Чтобы немного выиграть время, вы можете проиндексировать лишь первые несколько симво-

лов в значении столбца. Команда, приведенная ниже, создает индекс первых 16 символов в поле definition:

```
CREATE INDEX definition ON dictionary (id, definition(16));
```

Этот индекс значительно ускоряет поиск, когда слова ищутся по их полному определению, при этом он не требует значительного дискового пространства и процессорного времени.

Что касается соотношения времени запроса и размера базы данных (одного из основополагающих компромиссов в области проектирования баз данных), наиболее распространенная ошибка, особенно при проведении веб-скрапинга, когда приходится работать с большими объемами текстовых данных, – это хранение огромного объема повторяющихся данных. Например, вы хотите измерить частоту определенных фраз, упоминающихся на веб-сайтах. Эти фразы можно найти, воспользовавшись списком, или автоматически получить с помощью алгоритма анализа текста. Вы, возможно, хотели бы сохранить данные следующего вида:

```
+--------+-------------+------+-----+---------+----------------+
| Field  | Type        | Null | Key | Default | Extra          |
+--------+-------------+------+-----+---------+----------------+
| id     | int(11)     | NO   | PRI | NULL    | auto_increment |
| url    | varchar(200)| YES  |     | NULL    |                |
| phrase | varchar(200)| YES  |     | NULL    |                |
+--------+-------------+------+-----+---------+----------------+
```

Код, приведенный выше, добавляет в базу данных строку каждый раз, когда вы находите фразу на сайте, и записывает URL-адрес сайта, где эта фраза была найдена. Разбив данные на три отдельные таблицы, вы можете существенно уменьшить размер набора данных:

```
>DESCRIBE phrases
+--------+-------------+------+-----+---------+----------------+
| Field  | Type        | Null | Key | Default | Extra          |
+--------+-------------+------+-----+---------+----------------+
| id     | int(11)     | NO   | PRI | NULL    | auto_increment |
| phrase | varchar(200)| YES  |     | NULL    |                |
+--------+-------------+------+-----+---------+----------------+

>DESCRIBE urls
+-------+-------------+------+-----+---------+----------------+
| Field | Type        | Null | Key | Default | Extra          |
+-------+-------------+------+-----+---------+----------------+
| id    | int(11)     | NO   | PRI | NULL    | auto_increment |
| url   | varchar(200)| YES  |     | NULL    |                |
+-------+-------------+------+-----+---------+----------------+
```

```
>DESCRIBE foundInstances
+-------------+---------+------+-----+---------+----------------+
| Field       | Type    | Null | Key | Default | Extra          |
+-------------+---------+------+-----+---------+----------------+
| id          | int(11) | NO   | PRI | NULL    | auto_increment |
| urlId       | int(11) | YES  |     | NULL    |                |
| phraseId    | int(11) | YES  |     | NULL    |                |
| occurrences | int(11) | YES  |     | NULL    |                |
+-------------+---------+------+-----+---------+----------------+
```

Хотя определения таблиц стали больше, можно увидеть, что большинство столбцов – это просто целочисленные поля `id`. Они занимают гораздо меньше места. Кроме того, каждый URL-адрес и фраза сохраняются только один раз.

Если не установить некоторые пакеты или не вести подробные логи, невозможно будет понять, когда данные были добавлены, изменены или удалены из базы. В зависимости от доступного дискового пространства, частоты изменений и важности фиксирования внесенных изменений вы, возможно, захотите поставить временные метки: «created», «updated» и «deleted».

«Шесть шагов» в MySQL

В главе 3 я рассказала о проблеме «Шести шагов Википедии», когда нужно найти связь между любыми двумя статьями Википедии с помощью ряда ссылок (т. е. найти способ перейти от одной статьи Википедии к следующей по ссылке, ведущей с одной страницы на следующую).

Для того чтобы решить эту проблему, необходимо не просто создать ботов, которые могут не только просканировать сайт (что мы уже сделали), но и сохранить информацию надежным способом, чтобы в дальнейшем можно было легко выполнить анализ данных.

Автоинкрементные столбцы `id`, временные метки и таблицы – все они здесь вступают в игру. Чтобы выяснить, как лучше всего сохранить эту информацию, вы должны думать абстрактно. Ссылка – это то, что связывает страницу A со страницей B. Она может легко связать страницу B со страницей A, но это уже будет отдельное звено. Мы можем однозначно определить ссылку, сказав: «Имеется ссылка на странице A, которая связана со страницей B. То есть `INSERT INTO` links (fromPageId, toPageId) VALUES (A, B), где «A» и «B» – уникальные идентификаторы двух страниц.

Систему с двумя таблицами, предназначенную для хранения страниц и ссылок, наряду с датами создания и уникальными идентификаторами можно построить следующим образом:

```
CREATE TABLE `wikipedia`.`pages` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `url` VARCHAR(255) NOT NULL,
  `created` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`));

CREATE TABLE `wikipedia`.`links` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `fromPageId` INT NULL,
  `toPageId` INT NULL,
  `created` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`));
```

Обратите внимание, в отличие от предыдущих краулеров, которые печатают название страницы, я даже не сохраняю название страницы в таблице страниц. С чем это связано? Ну, запись названия страницы требует, чтобы вы фактически посетили страницу, чтобы извлечь ее. Если мы хотим написать эффективный веб-краулер, который заполнит эти таблицы, нам нужно сохранить страницу, а также ссылки на нее, даже если мы ее еще не посетили.

Конечно, преимущество ссылок и названий страниц Википедии заключается в том, что с помощью простых манипуляций можно легко перейти с одной страницы на другую, хотя этого нельзя сказать о всех сайтах. Например, *http://en.wikipedia.org/wiki/Monty_Python* указывает, что название страницы – «Monty Python».

Код, приведенный ниже, сохраняет все страницы Википедии, которые имеют «число Бэйкона» (количество ссылок между страницей и страницей про Кевина Бэйкона включительно) 6 или менее: `from urllib.request import urlopen`.

```
from bs4 import BeautifulSoup
import re
import pymysql

conn = pymysql.connect(host='127.0.0.1', unix_socket='/tmp/mysql.sock', user=
                       'root', passwd=None, db='mysql', charset='utf8')
cur = conn.cursor()
cur.execute("USE wikipedia")

def insertPageIfNotExists(url):
    cur.execute("SELECT * FROM pages WHERE url = %s", (url))
    if cur.rowcount == 0:
        cur.execute("INSERT INTO pages (url) VALUES (%s)", (url))
        conn.commit()
        return cur.lastrowid
    else:
```

```
        return cur.fetchone()[0]

def insertLink(fromPageId, toPageId):
    cur.execute("SELECT * FROM links WHERE fromPageId = %s AND toPageId = %s",
                (int(fromPageId), int(toPageId)))
    if cur.rowcount == 0:
        cur.execute("INSERT INTO links (fromPageId, toPageId) VALUES (%s, %s)",
                    (int(fromPageId), int(toPageId)))
        conn.commit()

pages = set()
def getLinks(pageUrl, recursionLevel):
    global pages
    if recursionLevel > 4:
        return;
    pageId = insertPageIfNotExists(pageUrl)
    html = urlopen("http://en.wikipedia.org"+pageUrl)
    bsObj = BeautifulSoup(html)
    for link in bsObj.findAll("a",
                              href=re.compile("^(/wiki/)((?!:).)*$")):
                              insertLink(pageId,
                                   insertPageIfNotExists(link.attrs['href']))
        if link.attrs['href'] not in pages:
            #Мы получили новую страницу, добавляем ее и ищем ссылки
            newPage = link.attrs['href']
            pages.add(newPage)
            getLinks(newPage, recursionLevel+1)
getLinks("/wiki/Kevin_Bacon", 0)
cur.close()
conn.close()
```

Рекурсию всегда сложно реализовать в коде, который планируется использовать в течение длительного времени. В данном случае переменная `recursionLevel` передается функции `getLinks`, которая с помощью этой переменной отслеживает глубину рекурсии (при каждом вызове функции `recursionLevel` увеличивается). Когда `recursionLevel` достигает 5, функция автоматически возвращается, прекращая поиск. Это ограничение гарантирует, что переполнение стека не произойдет.

Имейте в виду, что выполнение этой программы может занять несколько дней. Несмотря на то что я действительно уже запускала эту программу, сейчас моя база данных содержит лишь часть страниц с числом Кевина Бэйкона 6 или менее. Однако этого будет достаточно для нашего анализа расстояний между связанными статьями Википедии.

Дальнейшее рассмотрение этой проблемы и окончательное решение смотрите в главе 8, которая посвящена решению задач на направленных графах.

Электронная почта

Так же, как веб-страницы отправляются по протоколу HTTP, сообщения электронной почты отправляются по протоколу SMTP (Simple Mail Transfer Protocol). И так же, как вы используете технологию «клиент-сервер» для отправки веб-страниц по протоколу HTTP, серверы используют различных почтовых клиентов, например Sendmail, Postfix, или Mailman, для отправки и получения электронных писем.

Несмотря на то что отправить электронное письмо с помощью Python относительно легко, для этого требуется доступ к серверу SMTP. Настройка SMTP-клиента на вашем сервере или локальном компьютере сложна и выходит за рамки этой книги, но есть много отличных ресурсов, которые помогут решить эту задачу, особенно если вы работаете в Linux или Mac OS X.

В следующих примерах кода я буду исходить из того, что вы используете локальный SMTP-клиент. (Чтобы применить этот код для удаленного SMTP-клиента, просто замените `localhost` на адрес удаленного сервера.)

Для отправки сообщений электронной почты с помощью Python требуется всего девять строк кода:

```
import smtplib
from email.mime.text import MIMEText

msg = MIMEText("The body of the email is here")

msg['Subject'] = "An Email Alert"
msg['From'] = "ryan@pythonscraping.com"
msg['To'] = "webmaster@pythonscraping.com"

s = smtplib.SMTP('localhost')
s.send_message(msg)
s.quit()
```

Python предлагает два основных пакета для отправки сообщений электронной почты: *smtplib* и *email*.

Питоновский модуль email предлагает различные функции форматирования, позволяющие создавать «пакеты» сообщений для отправки. Объект `MimeText`, использованный в приведенном коде, создает пустое сообщение, подготовленное для передачи по низкоуровневому протоколу MIME (Multipurpose Internet Mail Extensions, или Многоцелевые расширения интернет-почты), с помощью которого осуществляются высокоуровневые SMTP-соединения. Объект MimeText `msg`

содержит входящие/исходящие электронные письма, а также тело и заголовок, который Python использует для создания правильно отформатированного электронного письма.

Пакет smtplib содержит информацию, требующуюся для подключения к серверу. Точно так же, как подключение к серверу MySQL, это подключение по завершении работы нужно отключить, чтобы не создавать слишком большого количества подключений.

Основное предназначение электронной почты можно расширить и сделать более полезным, если применить функцию:

```
import smtplib
from email.mime.text import MIMEText
from bs4 import BeautifulSoup
from urllib.request import urlopen
import time

def sendMail(subject, body):
    msg = MIMEText(body)
    msg['Subject'] = subject
    msg['From'] = "christmas_alerts@pythonscraping.com"
    msg['To'] = "ryan@pythonscraping.com"

s = smtplib.SMTP('localhost')
s.send_message(msg)
s.quit()

bsObj = BeautifulSoup(urlopen("https://isitchristmas.com/"))
while(bsObj.find("a", {"id":"answer"}).attrs['title'] == "NO"):
    print("It is not Christmas yet.")
    time.sleep(3600)
bsObj = BeautifulSoup(urlopen("https://isitchristmas.com/"))
sendMail("It's Christmas!",
         "According to http://itischristmas.com, it is Christmas!")
```

Данный скрипт один раз в час обращается к веб-сайту *https://isitchristmas.com* (главная особенность которого в том, что он выдает крупным шрифтом слово «ДА» или слово «НЕТ» в зависимости от дня года). Если он видит любую другую информацию, кроме «НЕТ», он отправляет вам по электронной почте уведомление о том, что сегодня Рождество.

Хотя данная программа кажется не намного полезнее календаря, висящего на стене, ее можно легко модифицировать, чтобы сделать массу крайне полезных вещей. Она может проинформировать вас по электронной почте о перебоях в работе сайта, неудачных тестах или появлении временно отсутствующего товара, заказанного на Amazon, – то, чего не может сделать ни один настенный календарь.

Глава 6

Чтение документов

Заманчиво думать об Интернете прежде всего как о хранилище текстовых сайтов, содержащих мультимедийный контент в новомодном формате Web 2.0, которым, как правило, можно пренебречь при проведении веб-скрапинга. Однако такое мнение игнорирует основополагающее определение Интернета как средства передачи файлов, не зависящего от типа контента.

Несмотря на то что начиная с конца 1960-х Интернет уже был в той или иной форме, формат HTML появился только после 1992 года. До этого момента Интернет главным образом включал в себя пересылку электронной почты и файлов. Веб-страниц в том виде, в каком мы знаем их теперь, на самом деле не было. Другими словами, Интернет не является хранилищем HTML-файлов. Это хранилище информации, где HTML-файлы часто используются в качестве средства ее визуального представления. Не имея возможности прочитать различные типы документов, включая текст, PDF, изображения, видео, электронные письма и т. д., мы теряем огромную часть имеющихся данных.

В этой главе освещается работа с документами, будь то скачивание документов в локальную папку или их чтение с последующим извлечением. Мы также расскажем о работе с различными типами кодировки текста, которые могут встретиться при работе с иноязычными HTML-страницами.

Кодировка документа

Кодировка документа сообщает приложению (будь то операционная система Вашего компьютера или ваш собственный код Python), как нужно прочитать документ. Эту кодировку обычно можно легко определить по расширению файла, хотя расширение файла не задает кодировку. Например, я могла бы без проблем сохранить *myImage.jpg* как *myImage.txt*, пока не попыталась бы открыть его с помощью текс-

тового редактора. К счастью, эта ситуация встречается редко, и расширение документа – это, как правило, все, что вам нужно знать, чтобы правильно прочитать документ.

На базовом уровне все документы кодируются нулевыми и единичными битами. Помимо того, есть алгоритмы кодирования, которые задают такие параметры, как «количество бит на символ» или «количество бит, выделенных для записи цвета одного пикселя» (при работе с графическими файлами). Кроме того, вы можете использовать алгоритм сжатия, как в случае с PNG-файлами.

Несмотря на то что работа с файлами, не относящимся к HTML, может поначалу напугать, будьте уверены, что с помощью соответствующей библиотеки Python прекрасно справится с любым форматом информации, который вы хотите скормить ему. Единственное различие между текстовым файлом, видеофайлом и файлом изображения заключается в том, как интерпретируются нулевые и единичные биты. В этой главе я расскажу о нескольких наиболее распространенных типах файлов: текстовых файлах, PDF, PNG, GIF.

Текст

Несколько необычно хранить файлы в Интернете в виде простого текста, но этот формат широко используется минималистскими или «олдскульными» сайтами, которые обладают огромными репозиториями текстовых файлов. Например, Internet Engineering Task Force (IETF) хранит все свои опубликованные документы в виде HTML, PDF и текстовых файлов (см. *http://bit.ly/1RCAj2f* в качестве примера). Большинство браузеров прекрасно отображает эти текстовые файлы, и вы можете без проблем собрать их.

Для большинства текстовых документов, например для файла с упражнением, расположенного по адресу http://www.pythonscraping.com/pages/warandpeace/chapter1.txt, вы можете использовать следующий метод:

```
from urllib.request import urlopen
textPage = urlopen(
                "http://www.pythonscraping.com/pages/warandpeace/chapter1.txt")
print(textPage.read())
```

Обычно, когда мы извлекаем страницу с помощью `urlopen`, мы превращаем ее в объект BeautifulSoup, чтобы выполнить парсинг HTML-структуры. В данном случае мы можем прочитать страницу

напрямую. Преобразование страницы в объект BeautifulSoup, хотя и вполне возможно, было бы контрпродуктивно – отсутствует необходимая для парсинга HTML-структура, поэтому библиотека здесь бесполезна. Прочитав текстовый файл как строку, вы просто должны проанализировать ее, как любую другую строку, прочитанную Python. Конечно, недостатком здесь является отсутствие возможности использовать HTML-теги в качестве контекстных подсказок, которые позволили бы вам отделить нужный текст от ненужного. Данный факт может представлять проблему, когда вы пытаетесь извлечь из текстовых файлов определенную информацию.

Кодировка текста и глобальный Интернет

Помните, я ранее говорила, что расширение файла – это все, что вам нужно для его правильного чтения? Ну, как ни странно, это правило не распространяется на основной формат документа *.txt*.

В девяти случаях из десяти считывание текста с использованием ранее описанных методов пройдет прекрасно. Однако работа с текстом в Интернете – дело непростое. Далее мы рассмотрим основы кодировок символов от ASCII до Unicode и расскажем, как с ними работать.

Краткий обзор типов кодировки

В начале 1990-х некоммерческая организация под названием Unicode Consortium предложила универсальную кодировку текста, разработав коды для символов, которые должны были использоваться во всех текстовых документах на всех языках. Цель состояла в том, чтобы включить в себя все символы, от букв латинского алфавита, на котором написана эта книга, до кириллицы, китайских иероглифов (象形), математических и логических символов (∑, ≥) и даже смайликов и символов «разное», например, знака биологической опасности (☣) и символа мира (☮).

Получившаяся в итоге кодировка, как вы, возможно, уже знаете, была названа UTF-8, что расшифровывается как «*Universal Character Set – Transformation Format 8 bit*» («Универсальный набор символов – Формат преобразования 8 бит»). Распространено заблуждение, что UTF-8 хранит все символы в 8 битах. Однако «8 бит» – это минимальное, а не максимальное количество бит, требующееся для вывода символа (если в UTF-8 для хранения каждого символа действительно использовалось бы 8 бит, это позволило бы использовать лишь 2^8,или 256, возможных символов. Очевидно, что этого было бы недостаточно для хранения остальных символов, начиная от китайских иероглифов и заканчивая символом мира).

На самом деле в кодировке UTF-8 каждый символ содержит индикатор, который говорит: «только один байт используется для кодирования этого символа» или «следующие два байта кодируют один символ», максимум можно использовать четыре байта. Поскольку эти четыре байта также содержат информацию о количестве байт, предназначенных для кодирования символа, полностью 32 бита (32 = 4 байт × 8 бит, поскольку 1 байт – это 8 бит) не используются. Действующий стандарт позволяет использовать до 21 бита информации, что в общей сложности составляет 2 097 152 возможных символов, из которых в настоящее время используются 1 114 112 символов.

Хотя Unicode стал настоящей находкой для многих приложений, привычки трудно менять, и кодировка ASCII по-прежнему популярна среди многих пользователей.

Стандарт кодировки ASCII, применяемый с 1960-х годов, использует 7 битов для кодирования каждого символа, что в общей сложности составляет 2^7, или 128, символов. Этого достаточно для латинского алфавита (как для прописных, так и строчных букв), знаков препинания и всех остальных символов, которые можно встретить на англоязычной клавиатуре.

Конечно, в 1960 году разница между текстовыми файлами, использующими 7-битную и 8-битную кодировки, была значительной, поскольку хранение информации было весьма дорогим. Программисты того времени спорили, что лучше – удобство работы с красивыми круглыми числами (требовалось добавить дополнительные биты) или практичность файла, занимающего меньше места на диске. В конце концов, 7-битовая кодировка победила. Однако в современных вычислениях каждая 7-битовая последовательность заполняется нулевым битом в начале[1], оставив нас в худшем из этих двух миров – размер файлов стал на 14% больше, а гибкость работы была ограничена всего 128 символами.

Создав UTF-8, разработчики решили использовать этот «бит заполнения» («padding bit») в документах с кодировкой ASCII, объявив, что все байты, начинающиеся с нулевого бита, являются однобайтовыми символами, и предложили две одинаковые схемы кодирования для ASCII и UTF-8. Таким образом, символы, приведенные ниже, можно использовать как в UTF-8, так и в ASCII:

[1] Этот «бит заполнения» еще напомнит о себе чуть позже, когда будем разбирать кодировку ISO.

```
01000001 - A
01000010 - B
01000011 - C
```

А следующие символы можно использовать только в кодировке UTF-8, и если документ интерпретируется как документ с кодировкой ASCII, они превратятся в непечатаемые символы:

```
11000011 10000000 - À
11000011 10011111 - ß
11000011 10100111 - ç
```

Кроме UTF-8, есть и другие стандарты UTF, например UTF-16, UTF-24, и UTF-32, хотя документы, закодированные в этих форматах, встречаются редко, за исключением необычных обстоятельств, которые выходят за рамки этой книги.

Конечно, проблема всех стандартов Unicode заключается в том, что любой документ на иностранном языке занимает гораздо больше места, чем должен. Несмотря на то что в вашем языке может использоваться только 100 символов или около того, вам понадобится, по крайней мере, 16 бит для каждого символа, а не просто 8 бит, как в случае с английской версией кодировки ASCII. Это увеличивает размер документов на иностранном языке примерно в два раза, по сравнению с текстовыми документами на английском языке, по крайней мере это верно для иностранных языков, не использующих латиницу.

ISO решает эту проблему путем создания отдельных кодировок для каждого языка. Как и Unicode, она использует те же самые коды ASCII, но «добавляет» нулевой бит в начале каждого символа, что позволяет создать специальные символы, требующиеся для различных языков. Лучше всего она подходит для европейских языков, которые также в значительной степени используют латинский алфавит (под который в кодировке отводится диапазон 0–127), но при этом требуют некоторых дополнительных специальных символов. Это позволяет кодировке ISO-8859-1 (разработанной для латинского алфавита) использовать такие символы, как дроби (например, ½) или знак авторского права (©).

Также используются и другие кодировки ISO, например ISO-8859-9 (турецкий), ISO 8859-2 (немецкий и другие языки) и ISO-8859-15 (французский и другие языки).

Хотя популярность документов, использующих кодировку ISO, сокращается в последние годы, около 9% сайтов в Интернете по-

прежнему предпочитают ее[1], поэтому важно разбираться в кодировках и уметь их определять перед проведением скрапинга сайта.

Кодировки в действии

В предыдущем разделе для функции `urlopen` мы использовали настройки по умолчанию, чтобы прочитать txt-документы, с которыми можем столкнуться в Интернете. Это подходит главным образом для работы с английским текстом. Однако при чтении текста на русском языке, арабском языке или просто встретив слово типа résumé, вы можете столкнуться с проблемами.

Возьмем следующий код:

```
from urllib.request import urlopen
textPage = urlopen(
            "http://www.pythonscraping.com/pages/warandpeace/chapter1-ru.txt")
print(textPage.read())
```

Этот код считывает первую главу исходного текста *«Войны и мира»* (написанного на русском и французском языках) и выводит на экран. Приведем часть этого текста:

```
b"\xd0\xa7\xd0\x90\xd0\xa1\xd0\xa2\xd0\xac \xd0\x9f\xd0\x95\xd0\xa0\xd0\x92\xd0\
x90\xd0\xaf\n\nI\n\n\xe2\x80\x94 Eh bien, mon prince.
```

Кроме того, открыв эту страницу в различных браузерах, мы видим бессмыслицу (рис. 6.1).

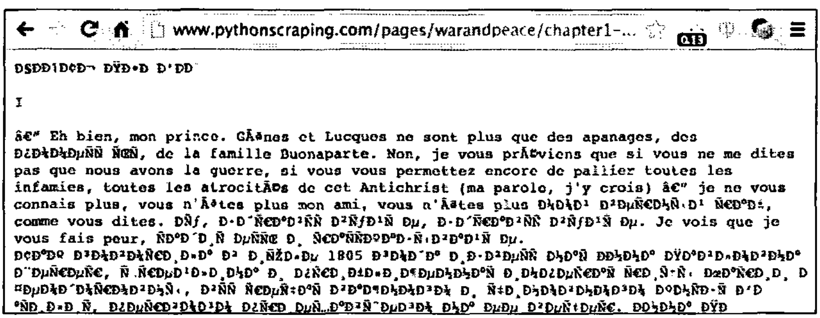


Рис. 6.1 ❖ Французский и кириллический текст, закодированный в ISO-8859-1 (кодировка текста, принятая по умолчанию во многих браузерах)

[1] Согласно сайту http://w3techs.com/technologies/history_overview/character_encoding, который использует веб-краулеры для сбора этих статистических данных.

Даже носителям русского языка этот текст будет трудно понять. Проблема в том, что Python пытается прочитать этот текст как документ в кодировке ASCII, тогда как браузер открывает его как документ в кодировке ISO-8859-1. Никто из них не понял, что речь идет о документе в кодировке UTF-8.

Мы можем явно задать строку в кодировке UTF-8, которая правильно отформатирует вывод кириллических символов:

```
from urllib.request import urlopen
textPage = urlopen(
            "http://www.pythonscraping.com/pages/warandpeace/chapter1-ru.txt")
print(str(textPage.read(), 'utf-8'))
```

Использование этого принципа в BeautifulSoup и Python 3.x выглядит так:

```
html = urlopen("http://en.wikipedia.org/wiki/Python_(programming_language)")
bsObj = BeautifulSoup(html)
content = bsObj.find("div", {"id":"mw-content-text"}).get_text()
content = bytes(content, "UTF-8")
content = content.decode("UTF-8")
```

Вы, возможно, захотите использовать во всех скраперах кодировку UTF-8. В конце концов, кодировка UTF-8 прекрасно понимает символы ASCII. Однако важно помнить, 9% веб-сайтов используют некоторые версии кодировки ISO. К сожалению, в случае с текстовыми документами невозможно конкретно определить используемую ими кодировку. Есть библиотеки, которые могут исследовать документ и выдвинуть наилучшее предположение (используя немного логики, чтобы понять, что «Ñ€Ð°ÑÑÐ°Ð°Ð·Ñ», вероятно, не является словом), которое часто бывает неверным.

К счастью, в случае с HTML-страницами кодировка, как правило, содержится в теге, расположенном в разделе <head> сайта. Большинство сайтов, особенно англоязычных, содержат тег:

```
<meta charset="utf-8" />
```

Тогда как веб-сайт European Computer Manufacturers Association содержит этот тег[1]:

```
<META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=iso-8859-1">
```

[1] ECMA была одним из первых разработчиков кодировки ISO, поэтому неудивительно, что ее веб-сайт использует эту кодировку.

Если вы планируете проводить масштабный веб-скрапинг, особенно скрапинг международных сайтов, целесообразно обратить внимание на этот мета-тег и использовать кодировку, которую он рекомендует для чтения этой страницы.

CSV

При проведении веб-скрапинга вы, вероятно, столкнетесь либо с CSV-файлом, либо с коллегой, который предпочитает данный формат. К счастью, Python предлагает великолепную библиотеку для чтения и записи файлов CSV. Хотя эта библиотека может обрабатывать различные варианты формата CSV, я главным образом сосредоточусь на стандартном формате. Если у вас особый случай (например, поля разделяются не запятыми, а специальными символами), обратитесь к документации!

Чтение CSV-файлов

Питоновская библиотека csv предназначена в первую очередь для работы с локальными файлами, основываясь на предположении, что нужные вам CSV-файлы хранятся на вашем компьютере. К сожалению, это не всегда так, особенно когда вы осуществляете веб-скрапинг. Есть несколько способов обойти это ограничение:

- скачайте файл вручную на локальный компьютер и укажите Python местоположение локального файла;
- напишите скрипт Python для скачивания файла, прочитайте файл и (по желанию) удалите его после извлечения;
- извлеките файл в виде строки из Интернета и оберните строку в объект `StringIO`, чтобы работать с ней как с файлом.

Несмотря на то что первые два варианта выполнимы, запись на жесткий диск файлов, которые можно легко обработать в оперативной памяти, является плохой практикой. Гораздо лучше прочитать файл в виде строки, обернуть строку в объект, который позволит обрабатывать ее как файл, не прибегая к сохранению файла. Скрипт, приведенный ниже, извлекает CSV-файл из Интернета (в данном случае список альбомов группы Monty Python по адресу *http://bit.ly/1QjuMv8*) и печатает его построчно в терминале:

```
from urllib.request import urlopen
from io import StringIO
import csv

data = urlopen("http://pythonscraping.com/files/MontyPythonAlbums.csv")
```

```
            .read().decode('ascii', 'ignore')
dataFile = StringIO(data)
csvReader = csv.reader(dataFile)

for row in csvReader:
print(row)
```

Вывод слишком большой, чтобы приводить его в полном объеме, но он должен выглядеть так:

```
['Name', 'Year']
["Monty Python's Flying Circus", '1970']
['Another Monty Python Record', '1971']
["Monty Python's Previous Record", '1972']
...
```

Как вы можете видеть из примера кода, читающий объект, возвращаемый `csv.reader` является итератором и состоит из питоновского списка объектов. Поэтому доступ к каждой строке объекта `csvReader` можно получить следующим образом:

```
for row in csvReader:
    print("The album \""+row[0]+"\" was released in "+str(row[1]))
```

Получаем вывод:

```
The album "Name" was released in Year
The album "Monty Python's Flying Circus" was released in 1970
The album "Another Monty Python Record" was released in 1971
The album "Monty Python's Previous Record" was released in 1972
...
```

Обратите внимание на первую строку: The album 'Name' was released in Year. Допустим, вы не хотите, чтобы эта информация попала в ваши данные, хотя ее можно легко исключить еще на стадии написания кода. Менее опытный программист может просто пропустить первую строку в объекте `csvReader` или написать специальный код для обработки этой строки. К счастью, есть альтернатива функции `csv.reader`, которая автоматически позаботится обо всем за вас. Введите `DictReader`:

```
from urllib.request import urlopen
from io import StringIO
import csv

data = urlopen("http://pythonscraping.com/files/MontyPythonAlbums.csv")
            .read().decode('ascii', 'ignore')
dataFile = StringIO(data)
dictReader = csv.DictReader(dataFile)
```

```
print(dictReader.fieldnames)

for row in dictReader:
print(row)
```

`csv.DictReader` возвращает значения каждой строки CSV-файла в виде словаря объектов (а не списка объектов). Названия полей хранятся в переменной `dictReader.fieldnames` и являются ключами словаря:

```
['Name', 'Year']
{'Name': "Monty Python's Flying Circus", 'Year': '1970'}
{'Name': 'Another Monty Python Record', 'Year': '1971'}
{'Name': "Monty Python's Previous Record", 'Year': '1972'}
```

Конечно, недостатком является то, что на создание, обработку и печать этих объектов `DictReaders`, в отличие от `csvReaders`, требуется немного больше времени, но удобство и практичность часто стоят дополнительных издержек.

PDF

Как пользователь Linux я знаю, как сложно работать с файлами *.docx*, используя немайкрософтовское программное обеспечение, и как трудно найти кодеки, которые могут интерпретировать новый формат медиафайлов Apple. В некотором смысле компания Adobe совершила революцию, разработав в 1993 году формат Portable Document Format. PDF-файлы позволили пользователям различных операционных систем просматривать графические и текстовые документы в едином универсальном формате.

Несмотря на то что хранение PDF-файлов в Интернете считается устаревшей практикой (зачем хранить контент в статичном, медленно загружающемся формате, когда его можно записать в формате HTML?), PDF-файлы встречаются повсеместно, особенно когда речь идет о работе с официальными бланками и документами.

В 2009 году британец Ник Иннес стал героем новостей, когда запросил в Бакингемширском городском совете информацию о результатах тестирования студентов, которая была опубликована в соответствии с Законом о свободе информации. После неоднократных просьб и отказов он, наконец, получил нужную ему информацию в виде 184 документов в формате PDF.

Хотя Иннес одержал победу и в конечном итоге получил правильно отформатированные данные, если бы он был экспертом по веб-

скрапингу, он, вероятно, мог бы сэкономить массу времени, потраченного в судах, обработав PDF-документы непосредственно с помощью одного из многочисленных питоновских модулей, предназначенных для парсинга PDF.

К сожалению, многие библиотеки парсинга PDF, разработанные для Python 2.x, не были обновлены с выпуском Python 3.x. Однако поскольку PDF – это относительно простой и открытый формат документа, существует много достойных питоновских библиотек, даже для Python 3.x, которые могут прочитать PDF-документы.

PDFMiner3K – одна из таких относительно легких в использовании библиотек. Она является очень гибкой, допуская использование командной строки или интеграцию в существующий код. Она также может обрабатывать различные языковые кодировки, что опять-таки пригодится при работе в Интернете.

Вы можете скачать модуль Python http://bit.ly/1FNj3Cb и установить его, распаковав папку и запустив:

```
$python setup.py install
```

Документация находится по адресу */pdfminer3k-1.3.0/docs/index.html* в распакованной папке. Однако текущая документация, как правило, в большей степени ориентирована на интерфейс командной строки, чем на интеграцию с кодом Python.

Ниже приводится базовая реализация кода, позволяющего считывать *любые* PDF-файлы в строку (на примере конкретного объекта, из которого читается PDF-файл):

```
from pdfminer.pdfinterp import PDFResourceManager, process_pdf
from pdfminer.converter import TextConverter
from pdfminer.layout import LAParams
from io import StringIO
from io import open

def readPDF(pdfFile):
    rsrcmgr = PDFResourceManager()
    retstr = StringIO()
    laparams = LAParams()
    device = TextConverter(rsrcmgr, retstr, laparams=laparams)

    process_pdf(rsrcmgr, device, pdfFile)
    device.close()

    content = retstr.getvalue()
    retstr.close()
    return content

pdfFile = urlopen("http://pythonscraping.com/pages/warandpeace/chapter1.pdf");
```

```
outputString = readPDF(pdfFile)
print(outputString)
pdfFile.close()
```

Замечательное свойство этой функции заключается в том, что если вы работаете с файлами локально, вы можете просто заменить обычный питоновский объект файла на объект, возвращаемый `urlopen`, и использовать следующую строку:

```
pdfFile = open("../pages/warandpeace/chapter1.pdf", 'rb')
```

Вывод не может быть идеальным, особенно когда речь идет о PDF-файлах с изображениями, замысловато отформатированным текстом или текстом, расположенным в таблицах или графиках. Однако вывод по итогам обработки PDF-файлов, содержащих только текст, не должен отличаться от вывода, полученного в результате обработки текстового файла.

Microsoft Word и .docx

Рискуя обидеть друзей из компании Microsoft, скажу: «Я не люблю Microsoft Word». Не потому, что это плохое программное обеспечение, а потому, что пользователи злоупотребляют этим форматом. Требуется особый талант, чтобы превратить простые текстовые документы или PDF-файлы в больших, неповоротливых, тяжело открываемых монстров, которые при передаче с одного компьютера на другой часто теряют форматирование, и по какой-то причине их содержимое меняется, хотя этого не должно происходить. Текстовые файлы никогда не были предназначены для частой передачи информации. Однако они широко используются на определенных сайтах, содержащих важные документы, информацию и даже графики и мультимедийный контент, в общем, все, что может и должно быть создано с помощью HTML.

Примерно до 2008 года продукты Microsoft Office использовали собственный файловый формат *.doc*. Этот бинарный формат файла было неудобно читать, и он плохо поддерживался другими текстовыми процессорами. Желая идти в ногу со временем и принять стандарт, который использовался бы другими программами, компания Microsoft решила использовать файловый формат Open Office XML, который поддерживается различными программами, в том числе программами с открытым кодом.

К сожалению, в Python поддержка этого файлового формата, используемого Google Docs, Open Office и Microsoft Office, по-прежнему

не велика. Существует библиотека python-docx, но она лишь дает пользователям возможность создавать документы и читать основную информацию о файле, например размер и название файла, а не фактическое содержание. Чтобы прочитать содержимое файла Microsoft Office, мы должны представить наше собственное решение.

Первым шагом является чтение XML из файла:

```
from zipfile import ZipFile
from urllib.request import urlopen
from io import BytesIO

wordFile = urlopen("http://pythonscraping.com/pages/AWordDocument.docx").
read()
wordFile = BytesIO(wordFile)
document = ZipFile(wordFile)
xml_content = document.read('word/document.xml')
print(xml_content.decode('utf-8'))
```

Этот код считывает удаленный документ Word как объект бинарного файла (объект `BytesIO` аналогичен объекту `StringIO`, ранее использованному в этой главе), распаковывает его, используя питоновскую стандартную библиотеку zipfile (все *.docx*-файлы заархивированы для экономии места), а затем считывает распакованные XML-файлы.

Документ Word, расположенный по адресу *http://bit.ly/1Jc8Zql*, показан на рис. 6.2.

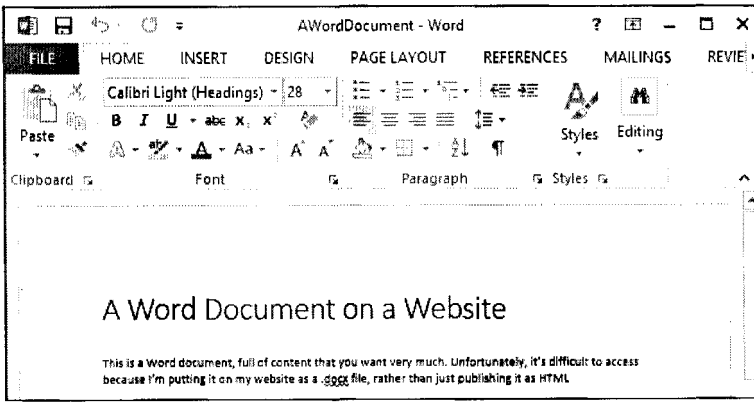


Рис. 6.2 ❖ Перед вами документ Word, он содержит нужный, но трудночитаемый контент, потому что я разместила его на своем сайте в виде файла .docx, вместо того чтобы опубликовать в формате HTML

Вывод скрипта Python, который считывает мой простой документ Word, выглядит следующим образом:

```
<!--?xml version="1.0" encoding="UTF-8" standalone="yes"?-->
<w:document mc:ignorable="w14 w15 wp14" xmlns:m="http://schemas.openx
mlformats.org/officeDocument/2006/math" xmlns:mc="http://schemas.open
xmlformats.org/markup-compatibility/2006" xmlns:o="urn:schemas-micros
oft-com:office:office" xmlns:r="http://schemas.openxmlformats.org/off
iceDocument/2006/relationships" xmlns:v="urn:schemas-microsoft-com:vm
l" xmlns:w="http://schemas.openxmlformats.org/wordprocessingml/2006/m
ain" xmlns:w10="urn:schemas-microsoft-com:office:word" xmlns:w14="htt
p://schemas.microsoft.com/office/word/2010/wordml" xmlns:w15="http://
schemas.microsoft.com/office/word/2012/wordml" xmlns:wne="http://sche
mas.microsoft.com/office/word/2006/wordml" xmlns:wp="http://schemas.o
penxmlformats.org/drawingml/2006/wordprocessingDrawing" xmlns:wp14="h
ttp://schemas.microsoft.com/office/word/2010/wordprocessingDrawing" x
mlns:wpc="http://schemas.microsoft.com/office/word/2010/wordprocessin
gCanvas" xmlns:wpg="http://schemas.microsoft.com/office/word/2010/wor
dprocessingGroup" xmlns:wpi="http://schemas.microsoft.com/office/word
/2010/wordprocessingInk" xmlns:wps="http://schemas.microsoft.com/offi
ce/word/2010/wordprocessingShape"><w:body><w:p w:rsidp="00764658" w:r
sidr="00764658" w:rsidrdefault="00764658"><w:ppr><w:pstyle w:val="Tit
le"></w:pstyle></w:ppr><w:r><w:t>A Word Document on a Website</w:t></
w:r><w:bookmarkstart w:id="0" w:name="_GoBack"></w:bookmarkstart><w:b
ookmarkend w:id="0"></w:bookmarkend></w:p><w:p w:rsidp="00764658" w:r
sidr="00764658" w:rsidrdefault="00764658"></w:p><w:p w:rsidp="0076465
8" w:rsidr="00764658" w:rsidrdefault="00764658" w:rsidrpr="00764658">
<w: r> <w:t>This is a Word document, full of content that you want ve
ry much. Unfortunately, it's difficult to access because I'm putting
it on my website as a .</w:t></w:r><w:prooferr w:type="spellStart"></
w:prooferr><w:r><w:t>docx</w:t></w:r><w:prooferr w:type="spellEnd"></
w:prooferr> <w:r> <w:t xml:space="preserve"> file, rather than just p
ublishing it as HTML</w:t> </w:r> </w:p> <w:sectpr w:rsidr="00764658"
 w:rsidrpr="00764658"> <w:pgszw:h="15840" w:w="12240"></w:pgsz><w:pgm
ar w:bottom="1440" w:footer="720" w:gutter="0" w:header="720" w:left=
"1440" w:right="1440" w:top="1440"></w:pgmar> <w:cols w:space="720"><
/w:cols&g; <w:docgrid w:linepitch="360"></w:docgrid> </w:sectpr> </w:
body> </w:document>
```

Здесь явно масса информации, но в ней невозможно разобраться. К счастью, весь текст документа, включая заголовок в верхней части, заключен в теги <w:t>, что облегчает сбор информации:

```
from zipfile import ZipFile
from urllib.request import urlopen
from io import BytesIO
from bs4 import BeautifulSoup
```

```
wordFile = urlopen("http://pythonscraping.com/pages/AWordDocument.docx").read()
wordFile = BytesIO(wordFile)
document = ZipFile(wordFile)
xml_content = document.read('word/document.xml')

wordObj = BeautifulSoup(xml_content.decode('utf-8'))
textStrings = wordObj.findAll("w:t")
for textElem in textStrings:
    print(textElem.text)
```

Вывод неидеален, но это лучше, чем ничего, а печать каждого тега <w:t> в новой строке позволяет легко увидеть разбивку текста в Word:

```
A Word Document on a Website
This is a Word document, full of content that you want very much. Unfortunately,
it's difficult to access because I'm putting it on my website as a .docx
 file, rather than just publishing it as HTML
```

Обратите внимание, что слово «docx» расположено в отдельной строке. В оригинальном XML оно заключено в тег <w:pstyle w:val="Title">. Речь идет о подчеркивании слова «docx» красной волнистой линией, применяемом Word. Word полагает, что при написании собственного же формата файлов была допущена орфографическая ошибка.

Заголовок документа предшествует тегу описания стиля <w:pstyle w:val="Title">. Хоть это особо и не поможет в идентификации заголовков (или другого текста), использование навигационных функций BeautifulSoup может быть полезным:

```
textStrings = wordObj.findAll("w:t")
for textElem in textStrings:
    closeTag = ""
    try:
        style = textElem.parent.previousSibling.find("w:pstyle")
        if style is not None and style["w:val"] == "Title":
        print("<h1>")
        closeTag = "</h1>"
    except AttributeError:
    #No tags to print
    pass
    print(textElem.text)
    print(closeTag)
```

Эту функцию можно легко дополнить, чтобы она печатала теги, в которые заключены различные стили текста, или помечала их каким-то другим способом.

ЧАСТЬ II

ПРОДВИНУТЫЙ СКРАПИНГ

Теперь, когда вы познакомились с некоторыми основами веб-скрапинга, начинается самое интересное. До этого момента ваши веб-скраперы были относительно бестолковыми. Они не в состоянии извлечь информацию, за исключением случаев, когда она размещена в удобном формате на сервере. Формы, интерактивный дизайн сайта и даже JavaScript могут поставить ваш скрапер в тупик. Короче говоря, они не годятся для извлечения информации, за исключением случаев, когда эта информация уже оптимизирована для извлечения.

Эта часть книги поможет вам проанализировать сырые данные, получить хранимую ими историю, которую сайты часто прячут под слоями JavaScript, формой входа и антискрапинговой защитой.

Вы узнаете, как использовать веб-скраперы для тестирования собственных сайтов, автоматизации процессов и доступа к Интернету в широком масштабе. По завершении этой части книги у вас появятся инструменты, которые позволят собрать практически любой тип данных, в любом формате, в любой части Интернета и работать с ним.

Глава 7

Очистка данных

До сих пор мы игнорировали проблему плохо отформатированных данных, работая с хорошо структурированной информацией и полностью игнорируя данные, которые не соответствовали нашим ожиданиям. Однако в ходе веб-скрапинга вы, как правило, не можете быть слишком придирчивы к извлекаемым данным.

Из-за пунктуационных ошибок, неправильного использования заглавных букв, разрывов строк и опечаток грязные данные являются большой проблемой Интернета. В этой главе я расскажу о нескольких инструментах и методах, которые помогут вам устранить первоисточник проблемы за счет изменения способа написания кода, и очистить информацию после ее попадания в базу данных.

Очистка данных на этапе создания кода

Точно так же, как вы пишите код для обработки исключений, вы должны использовать «оборонительную» тактику при разработке кода, чтобы справиться с различными неожиданностями.

В лингвистике *n-грамма* – это последовательность из *n* слов, используемых в тексте или речи. При выполнении естественной обработки языка она часто используется для разбивки текста, чтобы проанализировать часто используемые *n*-граммы или повторяющиеся наборы слов, которые неоднократно встречаются вместе.

В этом разделе мы сосредоточимся на получении правильно отформатированных *n*-грамм, а не на их использовании для проведения анализа. Позже, в главе 8, вы увидете, как можно использовать 2-граммы и 3-граммы для выполнения аннотирования и анализа текста.

Код, приведенный ниже, возвращает список 2-грамм, найденных в статье Википедии, посвященной языку программирования Python:

```
from urllib.request import urlopen
from bs4 import BeautifulSoup
```

```
def ngrams(input, n):
  input = input.split(' ')
  output = []
  for i in range(len(input)-n+1):
    output.append(input[i:i+n])
  return output

html = urlopen("http://en.wikipedia.org/wiki/Python_(programming_language)")
bsObj = BeautifulSoup(html)
content = bsObj.find("div", {"id":"mw-content-text"}).get_text()
ngrams = ngrams(content, 2)
print(ngrams)
print("2-grams count is: "+str(len(ngrams)))
```

Функция `ngrams` берет входную строку, разбивает ее на последовательность слов (предполагая, что все слова разделены пробелами) и добавляет в массив *n*-граммы (в данном случае 2-граммы).

Она возвращает несколько действительно интересных и важных 2-грамм из текста:

```
['of', 'free'], ['free', 'and'], ['and', 'open-source'], ['open-source', 'softwa
re']
```

но также и много мусора:

```
['software\nOutline\nSPDX\n\n\n\n\n\n\n\n\nOperating', 'system\nfamilies\n\n\n\n
AROS\nBSD\nDarwin\neCos\nFreeDOS\nGNU\nHaiku\nInferno\nLinux\nMach\nMINIX\nOpenS
olaris\nPlan'], ['system\nfamilies\n\n\n\nAROS\nBSD\nDarwin\neCos\nFreeDOS\nGNU\
nHaiku\nInferno\nLinux\nMach\nMINIX\nOpenSolaris\nPlan', '9\nReactOS\nTUD:OS\n\n
\n\n\n\n\n\n\nDevelopment\n\n\n\nBasic'], ['9\nReactOS\nTUD:OS\n\n\n\n\n\n\n\n\n
Development\n\n\n\nBasic', 'For']
```

Кроме того, поскольку для каждого встретившегося слова (за исключением последнего) создается 2-грамма, на момент написания книги эта статья будет содержать 7411 2-грамм. Не очень-то удобный для работы набор данных!

Используя некоторые регулярные выражения для удаления символов экранирования (например, \n) и фильтрацию для удалению любых символов Unicode, мы можем немного очистить вывод:

```
def ngrams(input, n):
    content = re.sub('\n+', " ", content)
    content = re.sub(' +', " ", content)
    content = bytes(content, "UTF-8")
    content = content.decode("ascii", "ignore")
    print(content)
    input = input.split(' ')
```

```
    output = []
    for i in range(len(input)-n+1):
        output.append(input[i:i+n])
    return output
```

Этот код сначала заменяет все экземпляры символа новой строки (или множественных символов новой строки) на пробел, затем заменяет все экземляры множественных пробелов в строке на одиночный пробел, гарантируя, что все слова отделены друг от друга одним пробелом. Символы экранирования устраняются путем кодировки контента в формат UTF-8.

Эти шаги значительно улучшают вывод функции, но есть еще некоторые проблемы:

```
['Pythoneers.[43][44]', 'Syntax'], ['7', '/'], ['/', '3'], ['3', '=='], ['==', '
2']
```

Теперь решения, требующиеся для обработки этих данных, становятся более интересными. Есть еще несколько правил, которые мы можем добавить, чтобы еще больше приблизиться к идеальным данным:

- удаляем «слова», состоящие из одного символа, за исключением случаев, когда этот символ – «i» или «a»;
- исключаем символы цитирования Википедии (числа в квадратных скобках);
- исключаем знаки (примечание: здесь это правило в некоторой степени упрощено и более подробно мы разберем его в главе 9, однако для этого примера оно прекрасно подходит).

Теперь, когда список «задач по уборке данных» стал длиннее, самое лучшее – это взять и поместить их в отдельную функцию `cleanInput`:

```
from urllib.request import urlopen
from bs4 import BeautifulSoup
import re
import string

def cleanInput(input):
    input = re.sub('\n+', " ", input)
    input = re.sub('\[[0-9]*\]', "", input)
    input = re.sub(' +', " ", input)
    input = bytes(input, "UTF-8")
    input = input.decode("ascii", "ignore")
    cleanInput = []
    input = input.split(' ')
    for item in input:
```

```
        item = item.strip(string.punctuation)
        if len(item) > 1 or (item.lower() == 'a' or item.lower() == 'i'):
            cleanInput.append(item)
    return cleanInput

def ngrams(input, n):
    input = cleanInput(input)
    output = []
    for i in range(len(input)-n+1):
        output.append(input[i:i+n])
    return output
```

Обратите внимание на использование `import string` и `string.punctuation`, чтобы получить список всех знаков препинания в Python. Вы можете посмотреть вывод `string.punctuation` в питоновском терминале:

```
>>> import string
>>> print(string.punctuation)
!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~
```

При использовании `item.strip (string.punctuation)` внутри цикла, перебирающего все слова в тексте, любые знаки препинания, находящиеся слева и справа от слова, будут удалены, при этом слова с дефисом (где знак пунктуации окружен с обеих сторон буквами) останутся нетронутыми.

Вывод этого кода дает нам более чистые 2-граммы:

```
['Linux', 'Foundation'], ['Foundation', 'Mozilla'], ['Mozilla', 'Foundation'], [
'Foundation', 'Open'], ['Open', 'Knowledge'], ['Knowledge', 'Foundation'], ['Fou
ndation', 'Open'], ['Open', 'Source']
```

Нормализация данных

Каждый сталкивался с плохо разработанной веб-формой: «Введите ваш номер телефона. Ваш номер телефона должен быть в формате xxx-xxx-xxxx». Как хороший программист вы, скорее всего, подумаете: «Почему бы им самостоятельно не выкинуть всё кроме цифр, а потом преобразовать результат в нужный формат?» Нормализация данных – это процесс, в ходе которого проверяется лингвистическая и логическая равнозначность строк, например телефонные номера (555) 123-4567 и 555.123.4567 отображаются или, по крайней мере, сравниваются как эквивалентные.

Используя код для получения *n*-грамм из предыдущего раздела, мы можем добавить некоторые функции нормализации данных.

Одна из очевидных проблем этого кода заключается в том, что он содержит много дублирующихся 2-грамм. Каждая найденная 2-грамма добавляется в список, при этом частота ее встречаемости не фиксируется. Нам не только более интересна частота встречаемости этих 2-грамм, чем просто их наличие, но и результаты применения алгоритмов очистки и нормализации данных. Если нормализация данных проходит успешно, общее количество уникальных *n*-грамм снизится, тогда как общее количество найденных *n*-грамм (то есть количество уникальных или неуникальных элементов, идентифицированных в качестве *n*-грамм) останется прежним. Другими словами, при том же самом количестве *n*-грамм у вас будет меньше «мусора».

К сожалению, в целях данного упражнения словари Python не отсортированы. «Сортировка словаря» здесь не имеет никакого смысла. Простым решением этой проблемы является `OrderedDict` из питоновского модуля collections:

```
from collections import OrderedDict

...

ngrams = ngrams(content, 2)
ngrams = OrderedDict(sorted(ngrams.items(), key=lambda t: t[1], reverse=True))
print(ngrams)
```

Здесь я воспользуюсь преимуществом питоновской функции `sorted` (https://docs.python.org/3/howto/sorting.html), чтобы поместить элементы в новый объект `OrderedDict`, отсортированный по значению. Вот результаты:

```
("['Software', 'Foundation']", 40), ("['Python', 'Software']", 38), ("['of', 'th
e']", 35), ("['Foundation', 'Retrieved']", 34), ("['of', 'Python']", 28), ("['in
', 'the']", 21), ("['van', 'Rossum']", 18)
```

На момент написания книги было найдено всего 7696 2-грамм, в их числе 6005 уникальных 2-грамм, самые популярные 2-граммы – «Software Foundation», затем «Python Software». Однако анализ результатов показал, что «Python Software» появляется как «Python software» еще два раза. Аналогично «van Rossum» и «Van Rossum» представлены в списке как две отдельные 2-граммы.

Добавление строки

```
input = input.upper()
```

в функцию `cleanInput` сохраняет неизменным общее количество 2-грамм (7696), тогда как количество уникальных *n*-грамм снижается до 5882.

Помимо этого, целесообразно остановиться и подумать, сколько вычислительных ресурсов вы хотите потратить на нормализацию данных. Есть ряд ситуаций, в которых различные варианты написания слов равнозначны, но для того, чтобы проанализировать эту равнозначность, вам нужно запустить проверку каждого отдельного слова и убедиться в том, что оно соответствует любому из заранее заданных правил эквивалентности.

Например, в списке встречается как 2-грамма «Python 1st», так и 2-грамма «Python first». Однако, универсальное правило, говорящее «Все слова «first», «second», «third» и т. д. обрабатывать как 1, 2, 3 и т. д. (или наоборот)», приведет к дополнительным 10 и более проверкам одного слова.

Аналогично непоследовательное использование дефисов (co-ordinated и coordi-nated), орфографические ошибки и прочие несоответствия повлияют на выделение *n*-грамм из текста и, возможно, исказят итоговые результаты, если этих ошибок будет довольно много.

Одним из решений при работе со словами, использующими дефис, могут быть полное удаление дефисов и обработка слова в виде отдельной строки, что требует выполнения только одной операции. Однако это означает, что фразы, использующие несколько дефисов подряд (all-too-common), будут обрабатываться как одно слово. Обработка дефисов как пробелов может быть наилучшим вариантом. Просто будьте готовы к тому, что в ваших данных будут случайно проскальзывать слова типа «co ordinated» и «ordinated attack»!

Очистка данных постфактум

В коде ваши возможности по очистке данных ограничены. Кроме того, вы можете столкнуться с набором данных, созданным другим пользователем, или набором данных, который становится настоящей проблемой, если не представляешь, как он изначально создавался, даже несмотря на то, что ты знаешь, как его чистить.

В такой ситуации немедленная реакция программистов – «написать скрипт», что может стать отличным решением. Однако есть программные инструменты сторонних производителей, например OpenRefine, которые могут не только быстро и легко очистить данные, но и позволяют людям, не искушенным в программировании, оперативно просматривать и использовать их.

OpenRefine

OpenRefine (http://openrefine.org/) – проект с открытым исходным кодом, запущенный компанией Metaweb в 2009 году. В 2010 году компания Google приобрела Metaweb, изменив название проекта с Freebase Gridworks на Google Refine. В 2012 году компания Google отказалась от поддержки Refine и снова изменила название, на OpenRefine, и с того момента каждый может внести вклад в развитие проекта.

Установка

Инструмент OpenRefine необычен тем, что хотя его интерфейс запускается в браузере, с технической точки зрения он является настольным приложением, которое нужно загрузить и установить. Вы можете скачать приложение для Linux, Windows и Mac OS X с вебсайта проекта.

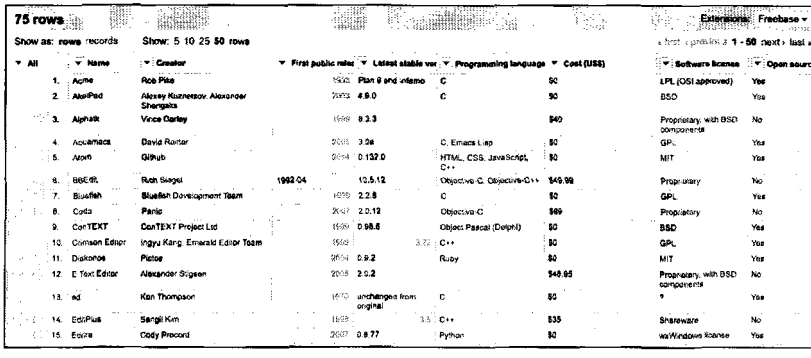
Обратите внимание

Если вы являетесь пользователем Mac и столкнулись с проблемой при открытии файла, перейдите в **System Preferences** (Системные настройки) → **Security & Privacy** (Безопасность и конфиденциальность) → **General** (Общие) и в пункте **Allow apps downloaded from** (Разрешить зазгрузку приложений из) выберите **Anywhere** (Любого источника). К сожалению, превратившись из проекта Google в проект с открытым исходным кодом, OpenRefine, похоже, потерял свою легитимность в глазах Apple.

Чтобы воспользоваться OpenRefine, сохраните ваши данные в формате CSV (вернитесь к разделу «Сохранение данных в формате CSV» в главе 5, если вам нужно освежить знания по этой теме). Кроме того, если у вас есть данные, хранящиеся в базе, вы можете экспортировать их в CSV-файл.

Использование OpenRefine

В следующих примерах мы будем использовать данные, извлеченные из таблицы Википедии Comparison of Text Editors (смотрите рис. 7.1). Хотя эта таблица относительно хорошо отформатирована, она содержит большое количество правок, внесенных пользователями в течение длительного времени, поэтому есть некоторые незначительные несоответствия. Кроме того, поскольку данные предназначены для пользователей, а не машин, некоторые варианты форматирования (например, «Free» вместо «$ 0.00») не подходят для использования в программном коде.

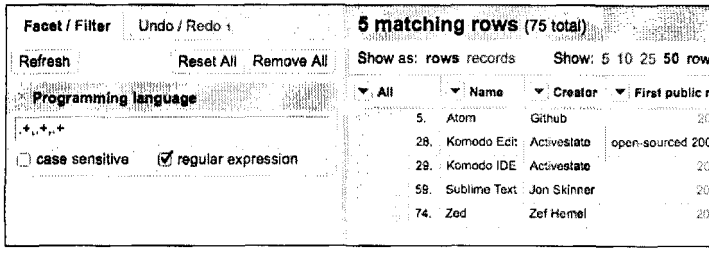


	Name	Creator	First public relea	Latest stable ver	Programming language	Cost (US$)	Software license	Open source
1.	Acme	Rob Pike	1993	Plan 9 and Inferno	C	$0	LPL (OSI approved)	Yes
2.	AlexPad	Alexey Kuznetsov, Alexander Shengalts	2005	4.9.0	C	$0	BSD	Yes
3.	Alpha9	Vince Darley	1999	8.3.3		$40	Proprietary with BSD components	No
4.	Aquamacs	David Reitter	2005	3.0a	C, Emacs Lisp	$0	GPL	Yes
5.	Atom	GitHub	2014	0.132.0	HTML, CSS, JavaScript, C++	$0	MIT	Yes
6.	BBEdit	Rich Siegel	1992-04	10.5.12	Objective-C, Objective-C++	$49.99	Proprietary	No
7.	Bluefish	Bluefish Development Team	1999	2.2.8	C	$0	GPL	Yes
8.	Coda	Panic	2007	2.0.12	Objective-C	$99	Proprietary	No
9.	ConTEXT	ConTEXT Project Ltd	1999	0.98.6	Object Pascal (Delphi)	$0	BSD	Yes
10.	Comson Editor	Ingyu Kang, Emerald Editor Team	1999	3.72	C++	$0	GPL	Yes
11.	Diakonos	Pistos	2004	0.9.2	Ruby	$0	MIT	Yes
12.	E Text Editor	Alexander Stigsen	2008	2.0.2		$48.95	Proprietary, with BSD components	No
13.	ed	Ken Thompson	1970	unchanges from original	C	$0	?	Yes
14.	EditPlus	Sergii Kim	1998	3.5	C++	$35	Shareware	No
15.	Editra	Cody Precord	2007	0.9.77	Python	$0	wxWindows license	Yes

Рис. 7.1 ❖ Данные из статьи Википедии «comparison of text editors», показанные в главном окне OpenRefine

Первое, что нужно отметить в отношении OpenRefine, – слева от каждого столбца находится символ стрелки. Стрелка раскрывает меню инструментов, которое можно использовать для фильтрации, сортировки, преобразования и удаления данных в этом столбце.

Фильтрация. Фильтрация данных выполняется с использованием двух методов: фильтров и фасетов. Фильтры используются регулярными выражениями для отбора данных. Например, «Покажите мне только те данные, которые содержат четыре или более языков программирования, разделенных запятыми, в столбце Programming Language», как показано на рис. 7.2.



	Name	Creator	First public re
5.	Atom	Github	20
28.	Komodo Edit	Activestate	open-sourced 200
29.	Komodo IDE	Activestate	20
59.	Sublime Text	Jon Skinner	20
74.	Zed	Zef Hemel	20

Рис. 7.2 ❖ Регулярное выражение «.+,.+,.+» отбирает значения, которые содержат, по крайней мере, три элемента, разделенных запятыми

Фильтры с легкостью можно объединять, редактировать и добавлять, управляя блоками в правом столбце. Кроме того, их можно объединить с фасетами.

Фасеты позволяют включать или исключать данные, комбинируя несколько критериев фильтрации (например, «Показать все строки, которые используют лицензию GPL или MIT и были впервые выпущены после 2005 года», как показано на рис. 7.3). Они являются встроенными средствами фильтрации. Например, фильтрация по числовому значению позволяет вам выбрать диапазон значений для включения.

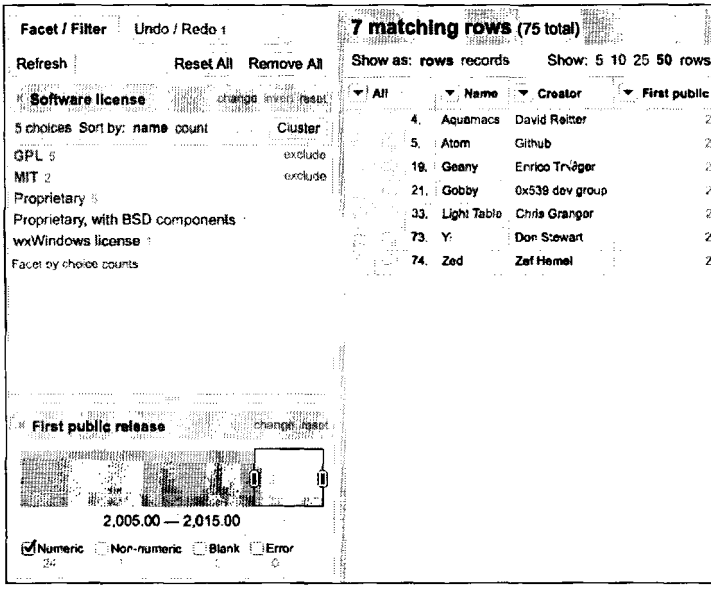


Рис. 7.3 ❖ Здесь показаны все текстовые редакторы, использующие лицензию GPL или MIT и впервые выпущенные после **2005** года

Отфильтровав данные, их можно в любой момент экспортировать в один из форматов, поддерживаемых OpenRefine (CSV, HTML или HTML-таблицы, Excel и некоторые другие).

Очистка. Фильтрация данных будет выполнена успешно только в том случае, если данные были изначально относительно чистыми. Например, в примере с фасетом, приведенном в предыдущем разделе, текстовый редактор с датой релиза **01-01-2006** не будет выбран в фасете **First public release** (Первый публичный релиз), который ищет значение **2006**, при этом игнорируя значения, выглядящие иначе.

Преобразование данных выполняется в OpenRefine с помощью языка выражений OpenRefine Expression Language или GREL (буква «G» досталась от предыдущего названия OpenRefine – Google Refine). Этот язык используется для создания компактных лямбда-функций, которые преобразуют значения в ячейках таблицы на основе простых правил. Например:

```
if(value.length() != 4, "invalid", value)
```

Когда эта функция применяется к столбцу **First public release** (Первый публичный релиз), он сохраняет значения ячеек, в которых дата записана в формате «YYYY», а все остальные столбцы помечает как недопустимые.

Нажав стрелку вниз рядом с меткой столбца и выбрав **edit cells** (редактировать ячейки) → **transform** (преобразовать), можно применить различные операторы GREL.

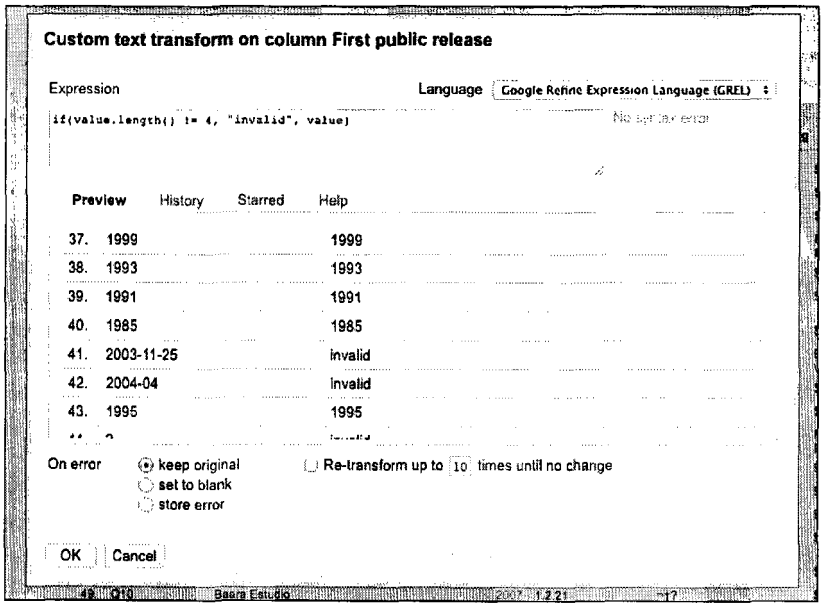


Рис. 7.4 ❖ Вставка оператора GREL в проект (показан оператор и предварительный результат его применения)

Однако объявление всех неидеальных значений недопустимыми хоть и облегчает их поиск, но не приносит особой пользы. По воз-

можности мы бы предпочли вытащить информацию из этих плохо отформатированных значений. Это можно сделать с помощью функции GREL `match`:

```
value.match(".*([0-9]{4}).*").get(0)
```

Она сопоставляет значение строки с заданным регулярным выражением. Если регулярное выражение соответствует строке, возвращается массив. Любые подстроки, которые соответствуют «группе захвата» в регулярном выражении (в выражении они разграничены скобками, в данном примере «[0-9] {4}»), возвращаются в виде значений массива.

По сути, этот код находит все последовательности из четырех цифр в строке и возвращает первую цифру. Этого, как правило, достаточно, чтобы извлечь годы из текста или из плохо отформатированных дат. Дополнительное преимущество заключается в том, что в случае несуществующих дат возвращается «null». (GREL не выдает исключения NullPointerException при выполнении операций с null переменной).

С помощью редактирования ячеек и GREL можно применить массу других преобразований данных. Полное руководство по языку можно найти на странице OpenRefine в GitHub.

Глава 8

Чтение и запись естественных языков

До сих пор данные, с которыми мы работали, были представлены, как правило, в формате чисел или исчисляемых значений. В большинстве случаев мы просто сохраняли данные без проведения какого-либо анализа постфактум. В этой главе мы попытаемся затронуть сложную тему под названием английский язык[1].

Как Google узнает, что нужно искать, когда вы набираете «cute kitten» в Картинках Google? Благодаря тексту, который сопровождает картинки с милыми котятами. Как YouTube узнает, что нужно возвратить определенное скетч-шоу группы Monty Python при вводе фразы «dead parrot» в строке поиска? Благодаря названию и тексту описания, которые сопровождают каждое загруженное видео.

На самом деле даже фраза «deceased bird monty python» немедленно возвратит то же самое скетч-шоу «Dead Parrot», хотя сама страница не содержит упоминаний слов «deceased» или «bird». Google знает, что «hot dog» – это еда, а не «сваренный в кипятке щенок». Каким образом? Все благодаря статистике!

Хотя вы, возможно, и не думаете, что анализ текста имеет что-то общее с вашим проектом, понимание его принципов может оказаться

---

[1] Хотя многие методы, описанные в этой главе, можно применить ко всем или большинству языков, мы сейчас рассмотрим обработку естественного языка на примере английского. Такие инструменты, как Natural Language Toolkit, сфокусированы на английском языке. 53,7% интернет-сайтов по-прежнему используют английский язык (на следующем месте – русский с всего лишь 6,3%, согласно http://w3techs.com/technologies/overview/content_language/all). Но кто знает? Лидерство английского языка в Интернете почти наверняка изменится в будущем и, возможно, в ближайшие годы нужно будет внести соответствующие изменения в программное обеспечение.

чрезвычайно полезным для проведения различных видов машинного обучения, а также дает более широкие возможности моделировать реальные задачи, используя теорию вероятности и математические алгоритмы.

Например, даже если аудиозапись содержит фоновый шум или искажения, музыкальный сервис Shazam сможет идентифицировать ее как определенную песню. Google выполняет автоматическое аннотирование изображений, используя только само изображение[1]. Например, сравнивая уже известные изображения хот-догов с другими изображениями хот-догов, поисковик может постепенно выяснить, как выглядит хот-дог, и выявить эти закономерности в других изображениях.

Аннотирование данных

В главе 7 мы рассмотрели разбивку текста на *n*-граммы или наборы фраз длиной *n* слов. На базовом уровне ее можно использовать, чтобы определить, какие наборы слов и фраз наиболее часто упоминаются в том или ином разделе текста. Кроме того, ее можно использовать для создания естественно звучащих аннотаций, обратившись к исходному тексту и отобрав предложения с наиболее популярными фразами.

Для аннотирования мы воспользуемся текстом, представляющим собой инаугурационную речь девятого президента США Уильяма Генри Гаррисона. Гаррисон установил два рекорда в истории Белого дома: одна из самых длинных инаугурационных речей и самый короткий срок президентства – 32 дня.

Для большей части примеров в этой главе мы используем в качестве источника полный текст этой речи.

Слегка изменив код, использованный в главе 7 для поиска *n*-грамм, мы получим код, который ищет наборы 2-грамм и сортирует их с помощью питоновской функции сортировки (модуль «operator»):

```
from urllib.request import urlopen
from bs4 import BeautifulSoup
import re
import string
import operator

def cleanInput(input):
    input = re.sub('\n+', " ", input).lower()
```

[1] См. статью «A Picture Is Worth a Thousand (Coherent) Words: Building a Natural Description of Images» от 17 ноября 2014 года (http://bit.ly/1HEJ8kX).

```
    input = re.sub('\[[0-9]*\]', "", input)
    input = re.sub(' +', " ", input)
    input = bytes(input, "UTF-8")
    input = input.decode("ascii", "ignore")
    cleanInput = []
    input = input.split(' ')
    for item in input:
        item = item.strip(string.punctuation)
        if len(item) > 1 or (item.lower() == 'a' or item.lower() == 'i'):
            cleanInput.append(item)
    return cleanInput

def ngrams(input, n):
    input = cleanInput(input)
    output = {}
    for i in range(len(input)-n+1):
        ngramTemp = " ".join(input[i:i+n])
        if ngramTemp not in output:
            output[ngramTemp] = 0
        output[ngramTemp] += 1
    return output

content = str(
      urlopen("http://pythonscraping.com/files/inaugurationSpeech.txt").read(),
              'utf-8')
ngrams = ngrams(content, 2)
sortedNGrams = sorted(ngrams.items(), key = operator.itemgetter(1),
reverse=True)
print(sortedNGrams)
```

Вывод выглядит следующим образом (приведен частично):

```
[('of the', 213), ('in the', 65), ('to the', 61), ('by the', 41), ('t
he constitution', 34), ('of our', 29), ('to be', 26), ('from the', 24
), ('the people', 24), ('and the', 23), ('it is', 23), ('that the', 2
3), ('of a', 22), ('of their', 19)
```

Из этих 2-грамм видно, что слово «constitution», похоже, является по праву самой популярной темой, вместе с тем различные «of the», «in the» и «to the» не представляют особой ценности. Как автоматически избавиться от ненужных слов максимально аккуратным способом?

К счастью, есть специалисты, которые тщательно изучают различия между «интересными» и «неинтересными» словами, и они могут помочь нам. Марк Дэвис, профессор лингвистики в Университете Бригама Янга, составляет Corpus of Contemporary American English (Корпус современного американского английского языка), коллек-

цию из более чем 450 миллионов слов, употреблявшихся за последнее десятилетие или около того в популярных американских изданиях.

Список 5000 наиболее часто встречающихся слов доступен бесплатно, и, к счастью, этого более чем достаточно, чтобы использовать в качестве основного фильтра для удаления наиболее распространенных 2-грамм. С добавлением функции isCommon первые 100 слов значительно улучшают результаты:

```
def isCommon(ngram):
    commonWords = ["the", "be", "and", "of", "a", "in", "to", "have", "it",
        "i", "that", "for", "you", "he", "with", "on", "do", "say", "this",
        "they", "is", "an", "at", "but","we", "his", "from", "that", "not",
        "by", "she", "or", "as", "what", "go", "their","can", "who", "get",
        "if", "would", "her", "all", "my", "make", "about", "know", "will",
        "as", "up", "one", "time", "has", "been", "there", "year", "so",
        "think", "when", "which", "them", "some", "me", "people", "take",
        "out", "into", "just", "see", "him", "your", "come", "could", "now",
        "than", "like", "other", "how", "then", "its", "our", "two", "more",
        "these", "want", "way", "look", "first", "also", "new", "because",
        "day", "more", "use", "no", "man", "find", "here", "thing", "give",
        "many", "well"]
    for word in ngram:
        if word in commonWords:
            return True
    return False
```

Этот код генерирует следующие 2-граммы, которые встречаются более 2 раз в теле текста:

```
('united states', 10), ('executive department', 4), ('general governm
ent', 4), ('called upon', 3), ('government should', 3), ('whole count
ry', 3), ('mr jefferson', 3), ('chief magistrate', 3), ('same causes'
, 3), ('legislative body', 3)
```

Первые два элемента в списке – это «United States» и «executive department», что вполне ожидаемо для инаугурационной речи президента.

Важно отметить, для фильтрации результатов мы используем список наиболее распространенных слов, относящихся к современному времени, что может быть неуместно, учитывая тот факт, что текст был написан в 1841 году. Однако, поскольку мы используем лишь первые 100 слов в списке, можно предположить, что эти слова регулярно употреблялись в течение долгого времени, в отличие от последних 100 слов. По всей видимости, мы получим удовлетворительные результаты, и можно не искать слова, которые наиболее часто употреб-

лялись в 1841 году, или составлять их список (несмотря на то что это может быть интересно).

Теперь, когда некоторые ключевые темы извлечены из текста, как это поможет нам составить аннотацию текста? Один из способов заключается в поиске первого предложения, которое содержит «популярную» *n*-грамму, согласно теории о том, что первое упоминание наиболее точно описывает тело контента. Первые пять наиболее популярных 2-грамм образуют следующий список:

- The Constitution of the United States is the instrument containing this grant of power to the several departments composing the government.
- Such a one was afforded by the executive department constituted by the Constitution.
- The general government has seized upon none of the reserved rights of the states.
- Called from a retirement which I had supposed was to continue for the residue of my life to fill the chief executive office of this great and free nation, I appear before you, fellow-citizens, to take the oaths which the constitution prescribes as a necessary qualification for the performance of its duties; and in obedience to a custom coeval with our government and what I believe to be your expectations I proceed to present to you a summary of the principles which will govern me in the discharge of the duties which I shall be called upon to perform.
- The presses in the necessary employment of the government should never be used to clear the guilty or to varnish crime.

Конечно, вряд ли это опубликуют в учебниках CliffsNotes[1] в ближайшее время, но, учитывая, что исходный документ состоял из 217 предложений в длину и четвертое предложение («Called from...») довольно хорошо раскрывает главную тему, это совсем не плохо для первого раза.

Марковские модели

Вы, возможно, слышали о генераторах текста на основе цепей Маркова. Они используются в развлекательных целях, например в приложении Twitov, а также для генерации спама, выглядящего как человеческий текст, чтобы одурачить системы обнаружения.

Все эти генераторы текста основаны на марковской модели, которая часто используется для анализа огромных наборов случайных со-

[1] CliffsNotes – популярная в США серия кратких пересказов литературных произведений для студентов. – *Прим. пер.*

бытий, где за одним дискретным событием следует другое дискретное событие с определенной вероятностью.

Например, мы можем построить систему наблюдения за погодой на основе марковской модели, как показано на рис. 8.1.

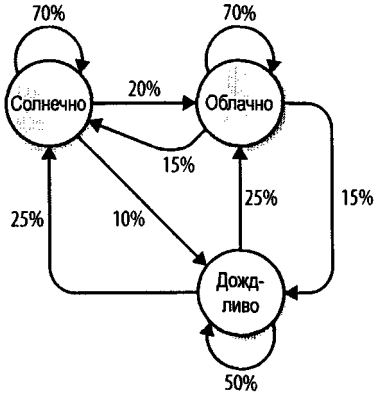


Рис. 8.1. Теоретическая система наблюдения за погодой на основе марковской модели

В этой модели вероятность того, что солнечный день сменится таким же солнечным днем, равна 70%, облачным днем – 20%, дождливым днем – 10%. Если день дождливый, существует 50%-ная вероятность того, что следующий день будет дождливым, 25%-ная вероятность солнечного дня, 25%-ная вероятность облачного дня.

Обратите внимание:

- все проценты, ведущие из каждого узла, должны складываться точно в 100%. Независимо от сложности модели вероятности перехода из каждого узла в другой должны составлять в итоге 100%;
- несмотря на то что есть только три возможных состояния погоды, вы можете использовать эту модель для генерирования любого списка состояний погоды;
- только состояние погоды в текущем узле определяет ваш маршрут. Если вы находитесь в узле «Солнечно», не имеет значения, какими были предыдущие 100 дней – солнечными или дождливыми, вероятность того, что следующий день будет солнечным, остается той же самой: 70%;
- до некоторых узлов будет сложнее добраться, чем до остальных. Стоящая за этим математика сложна, но довольно легко

увидеть, что «Дождливо» (судя по стрелкам, ведущим к этому узлу) является наименее вероятным состоянием в данной системе, чем «Солнечно» или «Облачно».

Очевидно, что перед нами очень простая система и модели Маркова могут быть любого размера. На самом деле, алгоритм Page Rank, используемый Google, частично основан на марковской модели, где веб-сайты представлены в виде узлов, а входящие/исходящие ссылки представлены в виде связей между узлами. «Вероятность» посещения данного узла представляет собой относительную популярность сайта. То есть если нашу систему наблюдения за погодой представить в виде Интернете, узел «Дождливо» будет иметь наименьший ранг страницы, в то время как узел «Облачно» будет иметь наибольший ранг страницы.

При всем этом давайте вернемся к более конкретному примеру: анализу и записи текста.

Снова используя инаугурационную речь Уильяма Генри Харрисона, проанализированную в предыдущем примере, мы можем написать код, который сгенерирует цепи Маркова произвольной длины (установим длину цепи равной 100) на основе текста:

```
from urllib.request import urlopen
from random import randint

def wordListSum(wordList):
    sum = 0
    for word, value in wordList.items():
        sum += value
    return sum

def retrieveRandomWord(wordList):

    randIndex = randint(1, wordListSum(wordList))
    for word, value in wordList.items():
        randIndex -= value
        if randIndex <= 0:
            return word

def buildWordDict(text):
    #Удаляем символы новой строки и кавычки
    text = text.replace("\n", " ");
    text = text.replace("\"", "");

    #Убедитесь, что знаки препинания обрабатываются как самостоятельные
    #"слова" таким образом они будут включены в марковскую це.ь
    punctuation = [',','.',';',':']
    for symbol in punctuation:
```

```
        text = text.replace(symbol, " "+symbol+" ");

    words = text.split(" ")
    #Удалите пустые слова
    words = [word for word in words if word != ""]

    wordDict = {}
    for i in range(1, len(words)):
        if words[i-1] not in wordDict:
                #Создаем новый словарь для этого слова
            wordDict[words[i-1]] = {}
        if words[i] not in wordDict[words[i-1]]:
            wordDict[words[i-1]][words[i]] = 0
        wordDict[words[i-1]][words[i]] = wordDict[words[i-1]][words[
                                                              i]] + 1

    return wordDict

text = str(urlopen("http://pythonscraping.com/files/inaugurationSpeech.txt")
          .read(), 'utf-8')
wordDict = buildWordDict(text)

#Генерируем цепь Маркова длиной 100
length = 100
chain = ""
currentWord = "I"
for i in range(0, length):
    chain += currentWord+" "
    currentWord = retrieveRandomWord(wordDict[currentWord])

print(chain)
```

Вывод этого кода будет меняться при каждом запуске, но в данном случае мы видим бессмысленный текст:

```
I sincerely believe in Chief Magistrate to make all necessary sacrifices and
oppression of the remedies which we may have occurred to me in the arrangement
and disbursement of the democratic claims them , consolatory to have been best
political power in fervently commending every other addition of legislation , by
the interests which violate that the Government would compare our aboriginal
neighbors the people to its accomplishment . The latter also susceptible of the
Constitution not much mischief , disputes have left to betray . The maxim which
may sometimes be an impartial and to prevent the adoption or
```

Так что же происходит в этом коде?

Функция `buildWordDict` в качестве аргумента берет строку текста, извлеченную из Интернета. Затем код выполняет некоторые процедуры очистки и форматирования текста, удаляет кавычки и ставит пробелы вокруг знака препинания, и таким образом знак препинания обрабатывается как отдельное слово. После этого он строит двумер-

ный словарь, так называемый «словарь словарей», который имеет следующий вид:

```
{word_a : {word_b : 2, word_c : 1, word_d : 1},
 word_e : {word_b : 5, word_d : 2},...}
```

В этом словаре «word_a» встретилось четыре раза, два раза после него следовало «word_b», один раз – «word_c» и один раз – «word_d». Слово «word_e» начинало семь словосочетаний, в пяти случаях после него следовало «word_b» и в двух случаях «word_d».

Если бы мы нарисовали модель для данного результата, из узла «word_a» исходила бы стрелка 50% к узлу «word_b» (поскольку в двух случаях из четырех после слова «word_a» следовало слово «word_b»), стрелка 25% к узлу «word_c» и стрелка 25% к узлу «word_d».

Создав словарь, его можно использовать как таблицу поиска, которая показывает маршрут передвижения, независимо от того, какое слово встретится в тексте[1]. Используя словарь словарей, мы можем оказаться в узле «word_e». Это означает, что мы передаем словарь `{word_b: 5, word_d: 2}` функции `retrieveRandomWord`. Эта функция, в свою очередь, извлекает случайное слово из словаря, взвешенное по частоте встречаемости.

Задав случайное начальное слово (в данном случае вездесущее «I»), мы можем легко перемещаться по марковской цепи, генерируя любое количество слов.

Шесть шагов Википедии: заключительная часть

В главе 3 мы создали скрапер, который собирает ссылки, переходя от одной статьи Википедии к другой (начиная от статьи про Кевина Бэйкона), и сохраняет их в базе данных. Почему мы снова вернулись к этой теме? Потому что проблема выбора маршрута, который начинается на странице о Кевине Бэйконе и заканчивается на странице об Эрике Айдле (т. е. расстояние между страницами *http://bit.ly/1d7SU7h* и *http://bit.ly/1GOSY7Z*), идентична поиску в цепи Маркова, где мы определяем первое и последнее слова. Подобные проблемы называют проблемами *направленных графов (directed graphs)*, где A → B необязательно означает, что B → A. После слова «football»

[1] *Исключением является последнее слово в тексте, поскольку после него ничего не следует.* В нашем примере последним словом является точка (.), которая удобна тем, что встречается в тексте еще 215 раз, и поэтому здесь не возникнет тупиковой ситуации. Однако при реальном использовании генератора Маркова вам, возможно, потребуется учитывать последнее слово.

может часто встретиться слово «player», но вы увидите, что после слова «player» слово «football» встречается гораздо реже. Несмотря на то что в статье о Кевине Бэйконе есть ссылка на статью про его родной город, Филадельфию, статья о Филадельфии не ссылается на статью о Кевине Бэйконе.

Наоборот, исходная игра «Шесть шагов до Кевина Бэйкона» – это проблема *ненаправленного графа (undirected graph)*. Если Кевин Бэйкон снялся в «*Коматозниках*» вместе с Джулией Робертс, то получается, что и Джулия Робертс снялась в «*Коматозниках*» с Кевином Бэйконом, поэтому связи распространяются в обоих направлениях (нет «направления»). Как правило, проблемы ненаправленных графов встречаются в информатике реже, чем проблемы направленных графов, и эти задачи довольно затратны с вычислительной точки зрения.

Хотя в плане решения этих проблем проделана большая работа, один из наилучших и наиболее распространенных способов найти кратчайший путь в направленном графе (и таким образом найти путь от статьи про Кевина Бэйкона до остальных статей Википедии) – это *поиск в ширину (breadth-first search)*.

Если выбран поиск в ширину, сначала исследуются ссылки на всех дочерних страницах 1-го уровня глубины, непосредственно связанных со стартовой страницей. Затем, если ссылка на интересующую страницу (страницу, которую вы ищите) не найдена, исследуются ссылки на всех дочерних страницах 2-го уровня глубины, связанных со страницами 1-го уровня глубины (которые в свою очередь связаны со стартовой страницей).

Комплексное решение для поиска в ширину с использованием таблицы ссылок, как описано в главе 5, выглядит следующим образом:

```
from urllib.request import urlopen
from bs4 import BeautifulSoup
import pymysql

conn = pymysql.connect(host='127.0.0.1', unix_socket='/tmp/mysql.sock',
                       user='root', passwd=None, db='mysql', charset='utf8')
cur = conn.cursor()
cur.execute("USE wikipedia")

class SolutionFound(RuntimeError):
    def __init__(self, message):
        self.message = message

def getLinks(fromPageId):
    cur.execute("SELECT toPageId FROM links WHERE fromPageId = %s", (fromPageId))
```

```
    if cur.rowcount == 0:
        return None
    else:
        return [x[0] for x in cur.fetchall()]

def constructDict(currentPageId):
    links = getLinks(currentPageId)
    if links:
        return dict(zip(links, [{}]*len(links)))
    return {}

#Дерево ссылок может быть пустым или содержать ссылки
def searchDepth(targetPageId, currentPageId, linkTree, depth):
    if depth == 0:
        #Останавливаем рекурсию и возвращаемся
        return linkTree
    if not linkTree:
        linkTree = constructDict(currentPageId)
        if not linkTree:
            #Ссылки не найдены. Продолжение в этом узле невозможно
            return {}
    if targetPageId in linkTree.keys():
        print("TARGET "+str(targetPageId)+" FOUND!")
        raise SolutionFound("PAGE: "+str(currentPageId))

    for branchKey, branchValue in linkTree.items():
        try:
            #Выполняем рекурсивный вызов для продолжения построения
            #дерева
            linkTree[branchKey] = searchDepth(targetPageId, branchKey,
                                              branchValue, depth-1)
        except SolutionFound as e:
            print(e.message)
            raise SolutionFound("PAGE: "+str(currentPageId))
    return linkTree

try:
    searchDepth(134951, 1, {}, 4)
    print("No solution found")
except SolutionFound as e:
    print(e.message)
```

Функции `getLinks` и `constructDict` являются вспомогательными. Они извлекают ссылки из страниц, хранящихся в базе данных, и преобразуют эти ссылки в словарь. Основная функция `searchDepth` работает рекурсивно, одновременно строит и ищет дерево ссылок, обра-

батывая один уровень за один проход. Она подчиняется следующим правилам:

- если заданный лимит рекурсии был достигнут (т. е. если она вызвала себя слишком много раз), функция возвращается без выполнения каких-либо операций;
- если полученный словарь ссылок пуст, пополняем его ссылками, расположенными на текущей странице. Если текущая страница не имеет ссылок, функция возвращается;
- если текущая страница содержит ссылку на интересующую нас страницу, функция выдает исключение о копировании себя выше по стеку, в котором было найдено решение. Затем каждый стек печатает текущую страницу и снова выдает исключение, в результате чего получаем идеальный список страниц, который приводит нас к решению на экране;
- если решение не найдено, функция вызывает себя, вычитая единицу из счетчика глубины для поиска следующего уровня ссылок.

Вывод по итогам поиска ссылки, связывающей страницу про Кевина Бэйкона (page ID 1 в моей базе данных) со страницей про Эрика Айдла (page ID 78520 в моей базе данных), выглядит так:

```
TARGET 134951 FOUND!
PAGE: 156224
PAGE: 155545
PAGE: 3
PAGE: 1
```

Получаем следующую взаимосвязь: Кевин Бэйкон → San Diego Comic-Con International → Брайан Фрауд → Терри Джонс → Эрик Айдл.

Кроме решения проблем «6 шагов» и моделирования частоты совместной встречаемости слов в предложениях, направленные и ненаправленные графы можно использовать для моделирования самых разнообразных ситуаций, встречающихся в веб-скрапинге. Как одни сайты связаны с другими сайтами? Какие научные статьи цитируют в других статьях? Какие продукты нужно размещать друг с другом на сайте? Какова сила этой взаимосвязи? Является ли ссылка взаимной?

Анализ этих базовых типов взаимосвязей может быть чрезвычайно полезен для построения моделей, визуализаций и прогнозов на основе собранных данных.

Natural Language Toolkit

До сих пор в этой главе мы были сосредоточены главным образом на статистическом анализе слов в тексте. Какие слова являются наиболее популярными? Какие слова являются необычными? Какие слова встретятся после интересующих нас слов? Как их сгруппировать вместе? Однако часто мы упускаем из виду, что же представляет из себя слово.

Natural Language Toolkit (NLTK) – это набор библиотек Python, предназначенный для поиска и тегирования частей речи в естественном (английском) тексте. Его разработка началась в 2000 году и в течение последних 15 лет десятки разработчиков по всему миру внесли свой вклад в этот проект. Хотя функциональность NLTK огромна (целые книги посвящены NLTK), в этом разделе основное внимание будет уделено лишь нескольким примерам его использования.

Установка и настройка

Модуль NLTK можно установить точно так же, как и любой другой модуль Python, либо непосредственно загрузив пакет с веб-сайта NLTK, либо используя инсталляторы сторонних разработчиков с ключевым словом «nltk». Подробные инструкции по установке смотрите на сайте NLTK.

После установки модуля скачайте его текстовые репозитории, таким образом знакомство с некоторыми возможностями NLTK будет более легким. Введите этот код в командной строке Python:

```
>>> import nltk
>>> nltk.download()
```

Он открывает загрузчик NLTK Downloader (рис. 8.2).

Я рекомендую установить все доступные пакеты. Поскольку все эти пакеты – на основе текста, они очень маленькие по размеру. Кроме того, никогда не знаешь, что в конечном итоге будешь использовать, а удалить пакеты можно довольно легко в любое время.

Статистический анализ с помощью NLTK

NLTK отлично подходит для получения статистической информации об общем количестве слов, количестве уникальных слов, частоте встречаемости слов в тех или иных разделах текста. Если все, что вам нужно, – это относительно простые вычисления (например, количество уникальных слов, используемых в определенном разделе

текста), импорт NLTK может быть излишним – это очень большой модуль. Однако если вам нужно выполнить относительно подробный анализ текста, в ваших руках – средство, позволяющее вычислить практически любую интересующую метрику.

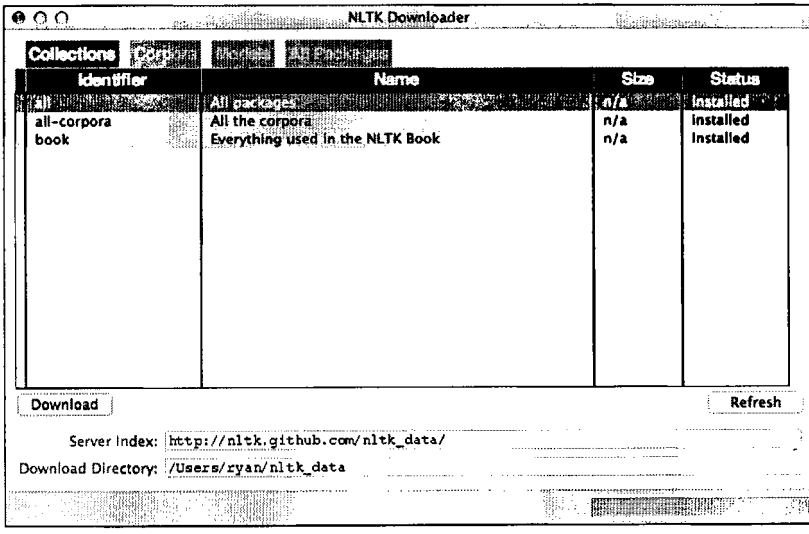


Рис. 8.2 ❖ NLTK Downloader позволяет просматривать и загружать дополнительные пакеты и текстовые библиотеки, связанные с модулем NLTK

Статистический анализ в NLTK всегда начинается с объекта `Text`. Объекты `Text` можно создать из простых питоновских строк следующим образом:

```
from nltk import word_tokenize
from nltk import Text

tokens = word_tokenize("Here is some not very interesting text")
text = Text(tokens)
```

Входным аргументом для функции `word_tokenize` может стать любой питоновский текст. Если вы считаете работу с длинными строками неудобной, но при этом хотите воспользоваться возможностями NLTK, есть довольно много книг, уже встроенных в библиотеки, к которым можно получить доступ с помощью функции `import`:

```
from nltk.book import *
```

Этот код загружает девять книг:

```
*** Introductory Examples for the NLTK Book ***
Loading text1, ..., text9 and sent1, ..., sent9
Type the name of the text or sentence to view it.
Type: 'texts()' or 'sents()' to list the materials.
text1: Moby Dick by Herman Melville 1851
text2: Sense and Sensibility by Jane Austen 1811
text3: The Book of Genesis
text4: Inaugural Address Corpus
text5: Chat Corpus
text6: Monty Python and the Holy Grail
text7: Wall Street Journal
text8: Personals Corpus
text9: The Man Who Was Thursday by G . K . Chesterton 1908
```

Во всех следующих примерах мы будем работать с text6 «Monty Python and the Holy Grail» (сценарий для фильма 1975 года).

Текстовыми объектами можно манипулировать так же, как и обычными питоновскими массивами, как если бы они были массивом, содержащим слова текста. Используя это свойство, можно подсчитать количество уникальных слов в тексте и сравнить его с общим количеством слов:

```
>>> len(text6)/len(words)
7.833333333333333
```

Код, приведенный выше, показывает, что каждое слово в скрипте было использовано в среднем примерно восемь раз. Кроме того, вы можете поместить текст в объект frequency distribution, чтобы посмотреть некоторые наиболее часто встречающиеся слова и частоты упоминаемости различных слов:

```
>>> from nltk import FreqDist
>>> fdist = FreqDist(text6)
>>> fdist.most_common(10)
[(':', 1197), ('.', 816), ('!', 801), (',', 731), ("'", 421), ('[', 3
19), (']', 312), ('the', 299), ('I', 255), ('ARTHUR', 225)]
>>> fdist["Grail"]
34
```

Поскольку это сценарий фильма, в выводе могут появиться некоторые артефакты. Например, часто появляется слово «ARTHUR», написанное заглавными буквами, потому что оно каждый раз предшествует прямой речи короля Артура в сценарии. Кроме того, перед каждой прямой речью ставится двоеточие (:), которое отделяет имя

персонажа от его прямой речи. Используя этот факт, мы можем увидеть, что в фильме 1197 строк с прямой речью!

То, что мы называли 2-граммами в предыдущих главах, NLTK называет биграммами (кроме того, вы можете услышать о 3-граммах, называемых «триграммами», но я лично предпочитаю говорить *n*-граммы вместо биграмм или триграмм). Вы можете легко создавать, искать и выводить 2-граммы:

```
>>> from nltk import bigrams
>>> bigrams = bigrams(text6)
>>> bigramsDist = FreqDist(bigrams)
>>> bigramDist[("Sir", "Robin")]
18
```

Чтобы найти 2-грамму «Sir Robin», мы должны сделать из нее массив («Sir», «Robin»), чтобы соответствовать способу представления 2-грамм во frequency distribution. Кроме того, есть модуль `trigrams`, который работает точно таким же образом. В общем случае вы можете также импортировать модуль `ngrams`:

```
>>> from nltk import ngrams
>>> fourgrams = ngrams(text6, 4)
>>> fourgramsDist = FreqDist(fourgrams)
>>> fourgramsDist[("father", "smelt", "of", "elderberries")]
1
```

Здесь функция `ngrams` вызывается для того, чтобы разбить текстовый объект на *n*-граммы определенной длины (длина регулируется вторым аргументом). В данном случае я разбиваю текст на 4-граммы. Затем я могу показать, что фраза «father smelt of elderbirries» встречается в сценарии ровно один раз.

Объекты frequency distribution, объекты text и *n*-граммы можно еще обрабатывать и применять в цикле. Код, приведенный ниже, распечатывает все 4-граммы, которые начинаются со слова «coconut»:

```
from nltk.book import *
from nltk import ngrams
fourgrams = ngrams(text6, 4)
for fourgram in fourgrams:
    if fourgram[0] == "coconut":
        print(fourgram)
```

Библиотека NLTK предлагает широкий спектр инструментов и объектов, предназначенных для управления, сортировки и анализа больших текстов. Хотя мы лишь коснулись некоторых моментов ее

использования, большинство из этих инструментов тщательно разработаны, и для людей, знакомых с Python, работа с ними интуитивно понятна.

Лексикографический анализ с помощью NLTK

До сих пор мы сравнивали и классифицировали все слова, исходя лишь из их собственных значений. При этом мы не делали различия между омонимами и не учитывали контекста, в котором эти слова употреблялись.

Хотя некоторые хотели бы признать проблему омонимов редкой, вы, возможно, удивитесь тому, насколько часто она возникает. Большинство носителей английского языка, вероятно, вообще не задумываются, что какое-то слово является омонимом, гораздо меньшее количество людей считают, что его можно спутать с другим словом, употребленном в ином контексте.

Предложение «He was objective in achieving his objective of writing an objective philosophy, primarily using verbs in the objective case» в плане синтаксического разбора не представляет труда для человека, однако веб-скрапер подумает, что одно и то же слово используется четыре раза, и просто проигнорирует информацию о значении каждого слова.

Кроме распознавания частей речи, важно определить контекст употребления слова. Например, вам нужно найти названия компаний, состоящих из часто употребляемых английских слов, или проанализировать мнения о компании «ACME Products is good» и «ACME Product is not bad», которые могут иметь одинаковый смысл, даже если одно предложение использует слово «good», а другое – «bad».

Набор тегов Penn Treebank

NLTK использует по умолчанию популярную систему тегирования частей речи Penn Treebank Project, разработанную в университете Пенсильвании. Хотя некоторые теги имеют вполне конкретный смысл (например, CC – это сочинительный союз), другие теги могут ввести в заблуждение (например, RP является частицей). Используйте следующие обозначения тегов:

CC	Coordinating conjunction (Сочинительный союз)
CD	Cardinal number (Количественное числительное)
DT	Determiner (Определитель)
EX	Existential *there* (Экзистенциальная конструкция *there*)
FW	Foreign word (Иностранное слово)
IN	Preposition, subordinating conjunction (Предлог, подчинительный союз)
JJ	Adjective (Прилагательное)
JJR	Adjective, comparative (Прилагательное, сравнительная степень)

JJS Adjective, superlative (Прилагательное, превосходная степень)
LS List item marker (Маркер списка)
MD Modal (Модальное слово)
NN Noun, singular or mass (Существительное в единственном числе или неисчисляемое существительное)
NNS Noun, plural (Существительное, множественное число)
NNP Proper noun, singular (Имя собственное, единственное число)
NNPS Proper noun, plural (Имя собственное, множественное число)
PDT Predeterminer (Предетерминатор или слово, предшествующее определяющему)
POS Possessive ending (Притяжательное окончание)
PRP Personal pronoun (Личное местоимение)
PRP$ Possessive pronoun (Притяжательное местоимение)
RB Adverb (Наречие)
RBR Adverb, comparative (Наречие, сравнительная степень)
RBS Adverb, superlative (Наречие, превосходная степень)
RP Particle (Частица)
SYM Symbol (Символ)
TO «to»
UH Interjection (Междометие)
VB Verb, base form (Глагол, основная форма)
VBD Verb, past tense (Глагол, прошедшее время)
VBG Verb, gerund or present participle (Глагол, герундий или причастие настоящего времени)
VBN Verb, past participle (Глагол, причастие прошедшего времени)
VBP Verb, non-third person singular present (Глагол, форма настоящего времени для всех лиц, кроме 3-го лица единственного числа)
VBZ Verb, third person singular present (Глагол, форма настоящего времени для 3-го лица единственного числа)
WDT Wh-determiner (Wh-определитель)
WP Wh-pronoun (Wh-местоимение)
WP$ Possessive wh-pronoun (Притяжательное wh-местоимение)
WRB Wh-adverb (Wh-наречие)

Кроме количественного анализа, NLTK позволяет определить смысл слов в зависимости от контекста (NLTK использует для этого собственные очень большие словари). На базовом уровне NLTK может определить части речи:

```
>>> from nltk.book import *
>>> from nltk import word_tokenize
>>> text = word_tokenize("Strange women lying in ponds distributing swords is no
basis for a system of government.  Supreme executive power derives from a mandate
from the masses, not from some farcical aquatic ceremony.")
>>> from nltk import pos_tag
>>> pos_tag(text)
[('Strange', 'NNP'), ('women', 'NNS'), ('lying', 'VBG'), ('in', 'IN')
, ('ponds', 'NNS'), ('distributing', 'VBG'), ('swords', 'NNS'), ('is'
, 'VBZ'), ('no', 'DT'), ('basis', 'NN'), ('for', 'IN'), ('a', 'DT'),
```

```
('system', 'NN'), ('of', 'IN'), ('government', 'NN'), ('.', '.'), ('S
upreme', 'NNP'), ('executive', 'NN'), ('power', 'NN'), ('derives', 'N
NS'), ('from', 'IN'), ('a', 'DT'), ('mandate', 'NN'), ('from', 'IN'),
 ('the', 'DT'), ('masses', 'NNS'), (',', ','), ('not', 'RB'), ('from'
, 'IN'), ('some', 'DT'), ('farcical', 'JJ'), ('aquatic', 'JJ'), ('cer
emony', 'NN'), ('.', '.')]
```

Каждое слово помещается в *кортеж*, содержащий слово и тег, определяющий часть речи (см. разметку «Penn Treebank» для получения дополнительной информации об этих тегах). Хотя задача определения частей речи может показаться простой, ее сложность становится очевидной в следующем примере:

```
>>> text = word_tokenize("The dust was thick so he had to dust")
>>> pos_tag(text)
[('The', 'DT'), ('dust', 'NN'), ('was', 'VBD'), ('thick', 'JJ'), ('so
', 'RB'), ('he', 'PRP'), ('had', 'VBD'), ('to', 'TO'), ('dust', 'VB')
]
```

Обратите внимание, что слово «dust» используется дважды в предложении: в качестве существительного и в качестве глагола. В обоих случаях NLTK определяет части речи правильно в зависимости от контекста употребления слов в предложении. NLTK определяет части речи, используя контекстно-свободную грамматику (context-free grammar) для английского языка. По сути, контекстно-свободные грамматики – наборы правил, которые задают порядок следования элементов в списках. В данном случае они задают порядок следования частей речи. Всякий раз, когда в тексте встречается двумысленное слово «dust», применяются правила контекстно-свободной грамматики и выбирается соответствующая часть речи, удовлетворяющая правилам.

Машинное обучение

С помощью пакета NLTK вы можете создать совершенно новые контекстно-свободные грамматики, например обучив его иностранному языку. Если вы размечаете вручную большие блоки текста, написанные на языке, который использует соответствующие теги Penn Treebank, вы можете загрузить эти тексты обратно в NLTK и обучить его правильно размечать новый текст. Эта операция является необходимым компонентом любого машинного обучения, которое мы рассмотрим в главе 11, в ней мы будем обучать скраперы распознавать символы CAPTCHA.

Итак, зачем нам знать, является ли слово глаголом или существительным в данном контексте? Это было бы интересно научно-исследовательской лаборатории информатики, но как это поможет нам в веб-скрапинге?

Очень распространенная проблема в веб-скрапинге – это поиск. Вы, возможно, извлекли текст с сайта, и нужно найти все экземпляры слова «google», но только те экземпляры, когда слово используется в качестве глагола, а не в качестве имени собственного. Или вы, возможно, ищете экземпляры слова «Google» (речь идет уже о компании) и в ходе поиска не хотите зависеть от правильности написания этого слова различными пользователями. Здесь функция `pos_tag` может быть чрезвычайно полезна:

```
from nltk import word_tokenize, sent_tokenize, pos_tag
sentences = sent_tokenize("Google is one of the best companies in the world.
I constantly google myself to see what I'm up to.")
nouns = ['NN', 'NNS', 'NNP', 'NNPS']

for sentence in sentences:
    if "google" in sentence.lower():
        taggedWords = pos_tag(word_tokenize(sentence))
            for word in taggedWords:
                if word[0].lower() == "google" and word[1] in nouns:
                    print(sentence)
```

Этот код печатает только те предложения, в которых слово «google» (или «Google») употребляется в качестве существительного, а не глагола. Конечно, вы можете конкретизировать поиск и запросить печать лишь тех экземпляров слова «Google», которые помечены как NNP (proper noun), но даже NLTK иногда совершает ошибки, и это даже неплохо в том смысле, что у вы вольны сами назначать часть речи в зависимости от контекста употребления слова.

Большинство случаев двусмысленности в естественном языке можно устранить с помощью функции `pos_tag`. Если вы будете искать в тексте не просто экземпляры интересующего вас слова (фразы), а экземпляры интересующего вас слова (фразы) *плюс* его тег, вы сможете значительно повысить точность и эффективность поиска при проведении веб-скрапинга.

Дополнительные ресурсы

Машинная обработка, анализ и интерпретация естественного языка являются одной из самых сложных задач в области компьютерных наук и бесконечное количество трудов и научных статей было написано на эту тему. Надеюсь, материал этой главы вдохновил вас выйти за рамки обычного понимания веб-скрапинга или, по крайней мере,

дал некоторое начальное направление, в котором нужно двигаться при реализации проекта, требующего анализа естественного языка.

Есть множество ресурсов с вводными курсами по обработке языка и работе с пакетом «Natural Language Toolkit». В частности, книга Стивена Берда, Эвана Клейна и Эдварда Лопера *Natural Language Processing with Python* представляет собой вводное и одновременно развернутое пособие по данной теме.

Книга Джеймса Пустейовски и Эмбер Стаббс *Natural Language Annotations for Machine Learning* является более развернутым руководством по теоретическим основам обработки естественного языка. Вам понадобится знание Python для выполнения уроков, в книге освещается работа с пакетом Natural Language Toolkit.

Глава 9

Краулинг сайтов, использующих веб-формы

Один из первых вопросов, с которым сталкиваешься, когда выходишь за рамки обычного веб-скрапинга, – «Как мне получить доступ к информации, которая скрыта за окном входа?» Веб все больше двигается в сторону интерактивности, социальных сетей и пользовательского контента. Веб-формы стали неотъемлемой частью этих сайтов и практически неизбежны. К счастью, работа с ними является относительно простой.

До этого момента большинство наших взаимодействий с веб-серверами состояло в использовании HTTP-запроса `GET`, чтобы получить нужную информацию. В этой главе мы сосредоточимся на методе `POST`, который передает информацию на веб-сервер для хранения и анализа.

В целом формы позволяют пользователям отправить `POST`-запрос, который интерпретируется и используется веб-сервером. Так же, как ссылочные теги помогают пользователям форматировать `GET`-запросы, HTML-формы помогают форматировать `POST`-запросы. С помощью небольшого программного кода можно легко создать эти запросы самостоятельно и отправить их с помощью скрапера.

Библиотека requests

Хотя перемещаться по веб-формам можно с помощью стандартных библиотек Python, иногда более удобный синтаксис упрощает жизнь. Когда вам нужно выполнить нечто большее, чем просто отправить базовый `GET`-запрос с помощью `urllib`, лучше не ограничиваться стандартными библиотеками Python.

Библиотека `requests` великолепно обрабатывает сложные HTTP-запросы, cookies, заголовки и многое другое.

Вот что говорит о стандартных инструментах Python разработчик библиотеки requests Кеннет Рейц:

«Стандартный модуль `urllib2` обеспечивает большую часть функциональных возможностей HTML, но его API никуда не годится. Он был разработан для другого времени, когда веб был другим. Он требует огромных трудозатрат при выполнении простейших задач (даже когда речь идет о подмене метода). Такого не должно быть. По крайней мере, не в Python».

Такого не должно быть. По крайней мере, не в Python.

Библиотеку requests, как и любую библиотеку Python, можно установить с помощью менеджера пакета (например, с помощью pip) или скачав и установив исходный файл.

Отправка простой формы

Большинство веб-форм состоит из нескольких HTML-полей, кнопки отправки и адреса страницы, где выполняется фактическая обработка формы. HTML-поля, как правило, состоят из текста, но могут также содержать поле **Upload file** (Загрузить файл) или какой-либо другой нетекстовый контент.

Большинство популярных веб-сайтов блокирует доступ к своим формам входа с помощью файла *robots.txt* (В приложении С обсуждаются правовые аспекты скрапинга таких сайтов). Поэтому я, не желая рисковать, разработала набор различных веб-форм на *pythonscraping.com*, на них вы можете протестировать ваши веб-скраперы. Простая форма находится по адресу http://bit.ly/1AGKPRU.

Полностью эта форма выглядит так:

```
<form method="post" action="processing.php">
First name: <input type="text" name="firstname"><br>
Last name: <input type="text" name="lastname"><br>
<input type="submit" value="Submit">
</form>
```

Здесь нужно отметить пару моментов: имена полей для ввода – `firstname` и `lastname`. Это важно. Имена этих полей определяют имена переменных-параметров, которые будут посланы в HTTP-запросе `POST` на сервер при отправке формы. Если вы хотите сымитировать обработку формы (отправив ваши собственные данные в HTTP-запросе `POST`), вы должны убедиться, что имена переменных совпадают с именами полей.

Второй момент, который нужно отметить, – обработка формы на самом деле происходит в *processing.php* (абсолютный путь *http://bit.ly/1d7TPVk*). Любые post-запросы к форме должны быть выполнены на *этой* странице, а не на странице, где размещена сама форма. Запомните: предназначение HTML-форм заключается лишь в том, чтобы помочь посетителям сайта правильно отформатировать запросы, отправляемые к странице, где происходит реальная обработка. Если вам не нужно форматировать сам запрос, то не надо беспокоиться и по поводу адреса страницы, где происходит обработка формы.

Отправка формы с помощью библиотеки requests выполняется с помощью четырех строк кода, включая импорт и печать контента (да, это так просто):

```
import requests

params = {'firstname': 'Ryan', 'lastname': 'Mitchell'}
r = requests.post("http://pythonscraping.com/files/processing.php", data=params)
print(r.text)
```

После отправки формы мы должны получить скрипт с контентом страницы:

```
Hello there, Ryan Mitchell!
```

Этот скрипт можно применить к большинству простых форм, встречающихся в Интернете. Например, форма подписки на новостную рассылку O'Reilly Media выглядит следующим образом:

```
<form action="http://post.oreilly.com/client/o/oreilly/forms/
            quicksignup.cgi" id="example_form2" method="POST">
    <input name="client_token" type="hidden" value="oreilly" />
    <input name="subscribe" type="hidden" value="optin" />
    <input name="success_url" type="hidden" value="http://oreilly.com/store/
            newsletter-thankyou.html" />
    <input name="error_url" type="hidden" value="http://oreilly.com/store/
            newsletter-signup-error.html" />
    <input name="topic_or_dod" type="hidden" value="1" />
    <input name="source" type="hidden" value="orm-home-t1-dotd" />
    <fieldset>
        <input class="email_address long" maxlength="200" name=
                "email_addr" size="25" type="text" value=
                "Enter your email here" />
        <button alt="Join" class="skinny" name="submit" onclick=
                "return addClickTracking('orm','ebook','rightrail','dod'
                                        );" value="submit">Join</button>
    </fieldset>
</form>
```

Хотя на первый взгляд этот код может показаться сложным, вспомните, что в большинстве случаев (мы рассмотрим исключения позже) вас интересуют лишь два момента:

- имя поля (или полей), которое используется при отправке данных (в данном случае название – `email_address`);
- атрибут action; то есть страница, где на самом деле происходит обработка формы (в данном случае http://post.oreilly.com/client/o/oreilly/forms/quicksignup.cgi).

Просто добавьте необходимую информацию и запустите его:

```
import requests
params = {'email_addr': 'ryan.e.mitchell@gmail.com'}
r = requests.post("http://post.oreilly.com/client/o/oreilly/forms/
                  quicksignup.cgi", data=params)
print(r.text)
```

В данном случае возвращенный сайт – это просто еще одна форма, заполнив которую, вы действительно подпишитесь на рассылку O'Reilly, тем не менее те же самые принципы, приведенные выше, можно с таким же успехом применить и к этой форме. Однако если вы хотите проделать аналогичные действия дома, я прошу вас использовать полученные знания с хорошими намерениями, а не закидывать издателя фальшивыми подписками.

Радиокнопки, флажки и другие элементы ввода данных

Очевидно, что не все веб-формы представляют собой набор текстовых полей с кнопкой отправки. Стандартная HTML-страница содержит разнообразные виды полей ввода: радиокнопки, флажки и поля выбора. В HTML5 добавлены ползунки выбора диапазона, адреса электронной почты, даты и многое другое. С пользовательскими полями JavaScript, полями выбора цвета, календарями и всем остальным, что только еще можно выдумать, возможности HTML становятся безграничны.

Несмотря на кажущуюся сложность того или иного поля, есть только две вещи, которые должны вас интересовать: название элемента и его значение. Название элемента можно легко определить, посмотрев на исходный код и отыскав атрибут `name`. Со значением иногда могут возникнуть сложности, поскольку непосредственно перед отправкой формы значение может быть заполнено JavaScript-кодом.

Поле выбора цвета, приведенное как пример достаточно экзотического вида поля, вероятно, будет иметь значение типа `#F03030`.

Если у вас нет информации о формате значения поля, есть целый ряд инструментов, которые можно использовать для отслеживания `GET` и `POST`- запросов, которые браузер отправляет на сайты и возвращает обратно. Как упоминалось ранее, самый лучший и, пожалуй, наиболее очевидный способ отслеживания `GET`-запросов – это просто посмотреть на URL-адрес сайта. Если URL-адрес выглядит примерно так:

```
http://domainname.com?thing1=foo&thing2=bar
```

знайте, что это соответствует форме следующего вида:

```
<form method="GET" action="someProcessor.php">
<input type="someCrazyInputType" name="thing1" value="foo" />
<input type="anotherCrazyInputType" name="thing2" value="bar" />
<input type="submit" value="Submit" />
</form>
```

что, в свою очередь, соответствует питоновскому словарю параметров:

```
{'thing1':'foo', 'thing2':'bar'}
```

Вы можете посмотреть его на рис. 9.1.

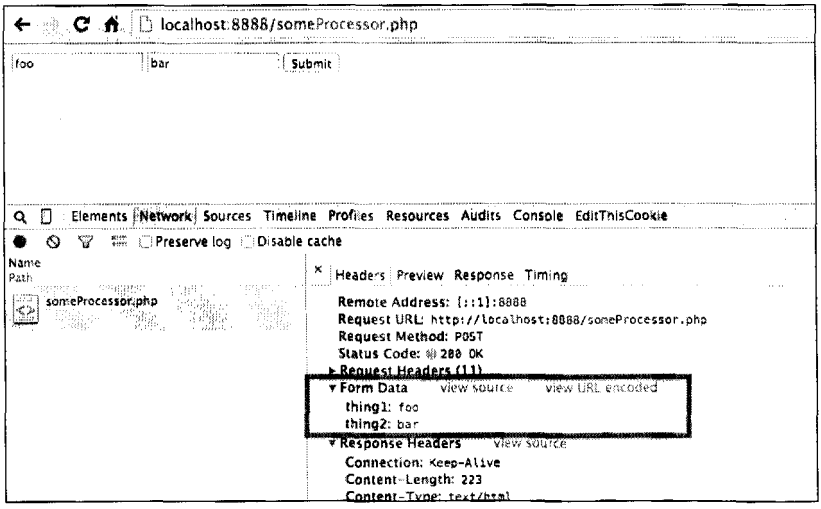


Рис. 9.1 ❖ Раздел Form Data (Данные формы), выделенный красной рамкой, показывает параметры POST-запроса «thing1» и «thing2» с их значениями «foo» и «bar»

Если вы столкнулись со сложным POST-запросом и хотите посмотреть, какие именно параметры ваш браузер посылает на сервер, самый простой способ посмотреть их – использовать инспектор браузера или панель разработчика браузера.

Инструменты разработчика Chrome можно открыть с помощью меню, выбрав **View** (Вид) → **Developer** (Разработчик) → **Developer Tools** (Инструменты разработчика). Инструменты разработчика выводят список всех запросов, которые ваш браузер генерирует, взаимодействуя с текущим сайтом, и позволяют детально посмотреть структуру этих запросов.

Отправка файлов и изображений

Хотя загрузка файлов широко распространена в Интернете, в вебскрапинге загрузка файлов используется нечасто. Однако вполне реальна ситуация, когда вы, возможно, захотите написать тест для вашего собственного сайта, который подразумевает загрузку файла. Во всяком случае, об этом полезно знать.

Форма загрузки файла расположена по адресу:

http://pythonscraping/files/form2.html

Форма, приведенная на странице, имеет следующую разметку:

```
<form action="processing2.php" method="post" enctype="multipart/form-data">
  Submit a jpg, png, or gif: <input type="file" name="image"><br>
  <input type="submit" value="Upload File">
</form>
```

За исключением тега <input> с атрибутом file, эта форма, по сути, выглядит точно так же, как текстовые формы, использованные в предыдущих примерах. К счастью, способы работы с формами загрузки файлов и текстовыми формами в питоновской библиотеке requests также очень схожи между собой:

```
import requests

files = {'uploadFile': open('../files/Python-logo.png', 'rb')}
r = requests.post("http://pythonscraping.com/pages/processing2.php",
                  files=files)
print(r.text)
```

Обратите внимание, что теперь не строка, а значение поля (с именем uploadFile) стало питоновским объектом File, возвращаемым функцией open. В этом примере я отправляю файл изображения, хранящийся на моем локальном компьютере по адресу *../files/Python-logo.png*, откуда питоновский скрипт и запускается.

Да, это действительно так просто!

Работа с логинами и cookies

До сих пор мы в основном рассматривали формы, которые позволяют вам отправить информацию на сайт или дают возможность просматривать необходимую информацию на странице сразу после заполнения формы. Чем эти формы отличаются от форм входа, которая позволяет вам оставаться на сайте в постоянно «залогиненном» состоянии в течение всего визита?

Большинство современных сайтов используют cookies, чтобы отслеживать, кто залогинился на сайте, а кто нет. Проверив подлинность ваших учетных данных, сайт сохраняет в вашем браузере маркер или cookie, который обычно содержит имя домена, с которого пришел маркер, срок действия маркера и информацию о тех частях дерева каталогов на сервере, которые могут использоваться маркером. Сайт использует этот cookie как своего рода доказательство подлинности, которое предъявляется каждой посещаемой странице во время посещения сайта. До середины 90-х годов, когда началось широкое использование cookies, у веб-сайтов были проблемы с надежной аутентификацией пользователей и отслеживанием их активности.

Хотя cookies являются отличным решением для веб-разработчиков, они могут представлять проблему для веб-скраперов. Отправив форму, важно отследить cookie, который форма отправила вам обратно, в противном случае следующая страница, которую вы посетите, будет выглядеть так, как будто вы вообще никогда не заходили на сайт.

Я создала простую форму входа по адресу *http://bit.ly/1KwvSSG* (имя пользователя может быть любым, но паролем должен быть «password»).

Эта форма обрабатывается по адресу *http://bit.ly/1d7U2I1* и содержит ссылку на страницу «главного сайта» *http://bit.ly/1JcansT*.

Если вы пытаетесь получить доступ к странице приветствия или странице профиля, не залогинившись, вы получите сообщение об ошибке и предупреждение о том, что вам нужно сначала указать имя и пароль. Для страницы профиля проверка учетных данных осуществляется с помощью cookies, заданных на странице входа и сохраненных в браузере.

С помощью библиотеки requests отследить cookies не составляет труда:

```
import requests

params = {'username': 'Ryan', 'password': 'password'}
r = requests.post("http://pythonscraping.com/pages/cookies/welcome.php", params)
print("Cookie is set to:")
print(r.cookies.get_dict())
print("----------")
print("Going to profile page...")
r = requests.get("http://pythonscraping.com/pages/cookies/profile.php",
                 cookies=r.cookies)
print(r.text)
```

В этом коде я отправляю параметры входа на страницу приветствия, которая выступает в качестве обработчика формы входа. Я получаю cookies из результатов последнего запроса, распечатываю результат для проверки, а затем отправляю их на страницу профиля, задав аргумент cookies.

Это хорошо работает в простых ситуациях, но что делать, если вы столкнулись с более сложным сайтом, который часто модифицирует cookies без предупреждения, или даже и не думали о работе с cookies? Функция session библиотеки request прекрасно справится в этом случае:

```
import requests

session = requests.Session()

params = {'username': 'username', 'password': 'password'}
s = session.post("http://pythonscraping.com/pages/cookies/welcome.php", params)
print("Cookie is set to:")
print(s.cookies.get_dict())
print("----------")
print("Going to profile page...")
s = session.get("http://pythonscraping.com/pages/cookies/profile.php")
print(s.text)
```

В данном случае объект session (полученный с помощью вызова requests.Session()) отслеживает информацию о сессии (cookies, заголовки и даже сведения о протоколах).

Requests – великолепная библиотека, на втором месте, возможно, только Selenium (о которой мы поговорим в главе 10), с точки зрения выполнения различных операций, не требующих привлечения программистов. Хотя идея о том, чтобы сесть, сложить руки и позволить библиотеке выполнить всю работу за вас выглядит очень заманчиво, крайне важно быть в курсе того, как выглядят cookies и как с ними работать при написании веб-скрапера. Это может сэкономить время,

которое уходит на болезненный процесс отладки, или позволит выяснить, почему сайт ведет себя странно!

Базовая HTTP-аутентификация

До появления cookies одним из популярных способов работы с логинами была *базовая HTTP-аутентификация* (*HTTP basic access authentification*). Вы по-прежнему можете встретить ее время от времени, особенно при работе с высокозащищенными или корпоративными сайтами, а также некоторыми API. Я создала страницу на *http://pythonscraping.com/pages/auth/login.php*, которая использует данный тип аутентификации (рис. 9.2).

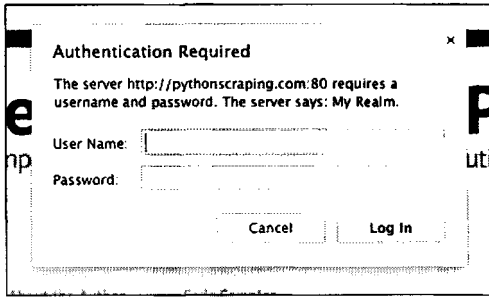


Рис. 9.2 ❖ Пользователь должен ввести имя и пароль, чтобы попасть на страницу, защищенную с помощью базовой аутентификации

Как и в примере выше, вы можете войти, используя любое имя пользователя, но паролем должно быть слово «password».

Пакет requests содержит модуль auth, специально предназначенный для работы с HTTP-аутентификацией:

```
import requests
from requests.auth import AuthBase
from requests.auth import HTTPBasicAuth

auth = HTTPBasicAuth('ryan', 'password')
r = requests.post(url="http://pythonscraping.com/pages/auth/login.php", auth=
                  auth)
print(r.text)
```

Хотя этот код похож на обычный POST-запрос, объект HTTPBasicAuth передается в запрос в качестве аргумента auth. Полученный в результате текст будет страницей, защищенной именем пользователя и паролем (или страницей Access Denied, если запрос отклонен).

Другие проблемы при работе с формами

Для вредоносных ботов веб-формы – главный объект атаки. Вам ведь не нужны боты, которые создают пользовательские аккаунты, снижают скорость обработки запросов сервером или размещают спам-комментарии в блогах? По этой причине многие современные сайты включают в свои HTML-формы различные функции безопасности, которые не всегда можно заметить.

Для получения справки по работе с CAPTCHA обратитесь к главе 11, в которой рассказывается об обработке изображений и распознавании текста в Python.

Если вы столкнулись с неидентифицируемой ошибкой либо сервер отказывается принять отправленную вами форму по неизвестной причине, обратитесь к главе 12, в которой рассказывается о ловушках, скрытых полях и других мерах безопасности, которые сайты предпринимают для того, чтобы защитить свои формы от ботов.

Глава 10

Скрапинг JavaScript-кода

Скриптовые клиентские языки – это языки, которые запускаются в самом браузере, а не на веб-сервере. Успешность выполнения кода, написанного на клиентском языке, зависит от способности Вашего браузера правильно интерпретировать и выполнить его. (Вот почему в вашем браузере всегда есть возможность отключить JavaScript.)

Частично из-за трудностей, связанных с поддержкой того или иного клиентского языка разработчиками браузера, количество клиентских языков существенно меньше количества серверных языков. Это хорошая новость для веб-скрапинга: чем меньше языков, с которыми приходится иметь дело, тем лучше.

По большей части, есть только два языка, с которыми вы будете часто сталкиваться в Интернете: ActionScript (который используется Flash-приложениями) и JavaScript. Сегодня ActionScript используется гораздо реже, чем это было 10 лет назад, и часто применяется для потоковой передачи мультимедийных файлов, в качестве платформы для онлайн-игр или для отображения страниц «Intro», о которых мы часто и представления не имеем, поскольку никто не хочет смотреть их. Во всяком случае, поскольку нет большого спроса на скрапинг Flash-страниц, я сосредоточилась на клиентском языке JavaScript, который повсеместно используется современными веб-страницами.

На сегодняшний день JavaScript является, безусловно, самым распространенным и наиболее хорошо поддерживаемым скриптовым клиентским языком в Интернете. Его можно использовать для сбора информации о перемещениях пользователей по сайту, отправки форм без перезагрузки страницы, встраивания мультимедийного контента и даже запуска целых онлайн-игр. Даже обманчиво простые на вид страницы могут часто содержать несколько кусков JavaScript-кода. Вы можете найти его внутри тегов `<script>` в исходном коде страницы:

```
<script>
    alert("This creates a pop-up using JavaScript");
</script>
```

Краткое введение в JavaScript

Наличие некоторого представления о том, что происходит в коде при проведении веб-скрапинга, может быть чрезвычайно полезным. Имейте это в виду, начав знакомство с языком JavaScript.

JavaScript является слабо типизированным языком с синтаксисом, который часто сравнивают с C ++ и Java. Хотя некоторые элементы синтаксиса, например операторы, циклы и массивы, могут быть схожи, слабая типизация и скриптовая природа языка могут представлять трудности для некоторых программистов.

Например, следующий код рекурсивно определяет числа Фибоначчи и печатает их в консоли разработчика браузера:

```
<script>
function fibonacci(a, b){
    var nextNum = a + b;
    console.log(nextNum+" is in the Fibonacci sequence");
    if(nextNum < 100){
        fibonacci(b, nextNum);
    }
}
fibonacci(1, 1);
</script>
```

Обратите внимание, что все переменные должны объявляться с помощью ключевого слова `var`, за которым следует имя переменной. Это похоже на знак `$` в PHP или описание типа (`int`, `String`, `List` и т. д.) в Java или C ++. Python необычен тем, что в нем не нужно объявлять переменные и явно указывать их тип.

Кроме того, JavaScript чрезвычайно удобен тем, что функции в нем можно передавать как переменные:

```
<script>
var fibonacci = function() {
    var a = 1;
    var b = 1;
    return function () {
        var temp = b;
        b = a + b;
        a = temp;
        return b;
```

```
    }
}
var fibInstance = fibonacci();
console.log(fibInstance()+" is in the Fibonacci sequence");
console.log(fibInstance()+" is in the Fibonacci sequence");
console.log(fibInstance()+" is in the Fibonacci sequence");
</script>
```

Это может показаться сложным на первый взгляд, но все становится понятным, если вы вспомните про лямбда-выражения (рассматриваются в главе 2). Переменная `fibonacci` определяется как функция. Значение функции возвращает функцию, которая печатает постоянно возрастающие числа Фибоначчи. При каждом своем вызове она возвращает функцию вычисления чисел Фибоначчи, которая выполняется снова и увеличивает значения.

Хотя это на первый взгляд может показаться запутанным, некоторые проблемы, например вычисление чисел Фибоначчи, как правило, решаются с помощью шаблонов, как было показано выше. Кроме того, передача функций в качестве переменных чрезвычайно полезна, когда речь идет об обработке действий пользователей и обратных вызовов, поэтому стоит освоить этот стиль программирования для чтения JavaScript.

Распространенные библиотеки JavaScript

Хотя знание ядра JavaScript важно, вы не сможете существенно продвинуться в своей работе с интернет-данными, если не будете использовать, по меньшей мере, одну из сторонних библиотек JavaScript. В исходном коде страницы можно встретить одну или несколько таких часто используемых библиотек.

Выполнение JavaScript с помощью Python довольно трудоемко и затратно с вычислительной точки зрения, особенно если вы делаете это в рамках масштабных проектов. Понимание основ работы с JavaScript и возможность проанализировать его непосредственно (без выполнения кода) могут быть крайне полезными и избавят вас от головной боли.

jQuery

jQuery является весьма распространенной библиотекой. 30% всех сайтов в Интернете и 70% самых популярных сайтов используют ее[1].

[1] Запись в блоге Дейва Метвина «The State of jQuery 2014» от 13 января 2014 года содержит более подробную статистику.

Сайт, использующий jQuery, легко определить, поскольку он содержит в своем коде подключение к jQuery, например:

```
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js"></
 script>
```

Если вы обнаружили, что сайт использует JQuery, вы должны быть осторожны при проведении скрапинга. JQuery используется для динамического создания HTML-контента, который появляется только после выполнения JavaScript. Если вы собираете контент страницы, используя традиционные методы, вы извлечете лишь предварительно загруженную страницу, которая появляется перед тем, как JavaScript создаст настоящее содержимое страницы (мы рассмотрим эту проблему скрапинга более подробно в разделе «Ajax и динамический HTML»).

Кроме того, скорее всего, эти страницы содержат анимацию, интерактивный контент и мультимедиа, которые могут усложнить скрапинг.

Google Analytics

50% всех сайтов используют Google Analytics[1], данный факт делает ее, пожалуй, самой популярной библиотекой JavaScript и наиболее популярным инструментом отслеживания поведения интернет-пользователей. Кстати, оба сайта – *http://pythonscraping.com* и *http://www.oreilly.com/* – используют Google Analytics.

Довольно легко определить, использует ли страница Google Analytics. В нижней части страницы будет размещен JavaScript-код, аналогичный приведенному ниже (взято с сайта O'Reilly Media):

```
<!-- Google Analytics -->
<script type="text/javascript">

var _gaq = _gaq || [];
_gaq.push(['_setAccount', 'UA-4591498-1']);
_gaq.push(['_setDomainName', 'oreilly.com']);
_gaq.push(['_addIgnoredRef', 'oreilly.com']);
_gaq.push(['_setSiteSpeedSampleRate', 50]);
_gaq.push(['_trackPageview']);

(function() { var ga = document.createElement('script'); ga.type =
'text/javascript'; ga.async = true; ga.src = ('https:' ==
```

[1] W3Techs, «Usage Statistics and Market Share of Google Analytics for Websites» (*http://w3techs.com/technologies/details/ta-googleanalytics/all/all*).

```
document.location.protocol ? 'https://ssl' : 'http://www') +
'.google-analytics.com/ga.js'; var s =
document.getElementsByTagName('script')[0];
s.parentNode.insertBefore(ga, s); })();

</script>
```

Этот скрипт вызывает код отслеживания Google Analytics (когда пользователь открывает страницу сайта), специальные cookies, используемые для отслеживания перемещений по страницам. Иногда он может стать проблемой для веб-скраперов, которые предназначены для выполнения JavaScript-кода и обработки cookies (например, для скраперов, которые используют библиотеку Selenium, обсуждается далее в этой главе).

Если сайт использует Google Analytics или аналогичную систему веб-аналитики и вы не хотите, чтобы ваши действия по сбору информации с сайта были обнаружены, убедитесь в том, что вы отказались от любых cookies, используемых для веб-аналитики, или вообще откажитесь от использования cookies.

Google Maps

Работая в Интернете, вы почти наверняка видели приложение Google Maps, встроенное в веб-сайт. Его интерфейс позволяет легко встроить карты с пользовательской информацией о любом сайте.

Если вы собираете какие-либо географические данные, то понимание механизма работы Google Maps позволит легко получить координаты *широта/долгота* и даже точный адрес интересующего вас места. Один из наиболее распространенных способов обозначить местоположение в Google Maps – использование *маркера* (также известного как булавка).

Маркеры можно вставить в любую Google-карту, воспользовавшись кодом, например вот таким:

```
var marker = new google.maps.Marker({
      position: new google.maps.LatLng(-25.363882,131.044922),
      map: map,
      title: 'Some marker text'
  });
```

Python позволяет легко извлечь все, что находится по этим координатам, и вывести список координат широта/долгота.

Используя «обратного геокодирование», вы можете перевести эти координаты в адреса, которые удобны в плане хранения и анализа.

Ajax и динамический HTML

До сих пор единственным способом взаимодействия с веб-сервером была отправка HTTP-запроса через обновление страницы. Если вы когда-либо отправляли форму или извлекали информацию с сервера без перезагрузки страницы, вы, вероятно, использовали сайт с технологией *Ajax*.

Вопреки распространенному мнению, Ajax – это не язык, а набор технологий для выполнения определенной задачи (так же, как и веб-скрапинг). Ajax означает Asynchronous JavaScript and XML (Асинхронный JavaScript и XML) и используется для передачи информации на сервер и получения от него ответа без выполнения отдельного запроса страницы. Примечание: никогда не говорите: «Этот сайт написан на Ajax». Будет правильнее сказать: «Эта форма использует технологию Ajax для взаимодействия с веб-сервером».

Как и Ajax, *динамический HTML (dynamic HTML)*, или DHTML, представляет собой набор технологий, используемых для решения общей задачи. DHTML – это сочетание HTML-кода, языка CSS и клиентского скриптового языка, которое используется для изменения HTML-элементов на странице. Кнопка может появиться только после того, как пользователь переместит курсор, цвет фона может измениться по щелчку мыши, или Ajax-запрос вызовет новый блок контента для загрузки.

Обратите внимание, что хотя слово «динамический» обычно ассоциируется с такими словами, как «перемещение» или «изменение», наличие интерактивных HTML-компонентов, динамичных изображений или встроенного медиаконтента необязательно делает страницу DHTML, хотя она может выглядеть как динамическая. Кроме того, на некоторых наиболее скучных, статичных на вид интернет-страницах могут быть запущены скрытые DHTML-процессы в зависимости от использования JavaScript для работы с HTML и CSS.

Собирая информацию с большого количества различных сайтов, вы рано или поздно столкнетесь с ситуацией, когда контент страницы, отображаемый в браузере, не соответствует контенту, который вы видите в извлекаемом исходном коде страницы. вы можете посмотреть результат работы скрапера и лишь почесать затылок, пытаясь выяснить, куда пропала вся та информация, которую вы только что видели в браузере.

Кроме того, у веб-страницы может быть страница загрузки, которая, похоже, перенаправляет вас на другую страницу, но при этом вы

можете заметить, что во время этого перенаправления URL-адрес страницы никогда не меняется.

Обе ситуации вызваны тем, что ваш скрапер не способен выполнить JavaScript, который творит волшебство со страницей. Если JavaScript не был запущен, HTML-код остается неизменным, и поэтому извлеченная страница может серьезно отличаться от той, что вы видите в вашем браузере, выполняющем JavaScript без проблем.

Есть несколько способов, благодаря которым страница может использовать Ajax или DHTML для изменения/загрузки контента, но в подобных ситуациях есть только два решения: напрямую собрать контент из JavaScript или использовать пакеты Python, способные выполнить сам JavaScript и собрать информацию с сайта именно в том виде, в каком она отображается в вашем браузере.

Выполнение JavaScript в Python с помощью библиотеки Selenium

Selenium – мощный инструмент веб-скрапинга, разработанный изначально для тестирования веб-сайтов. Сейчас он также используется, когда нужно составить точный «портрет» сайта (посмотреть, как он отображается в браузере). Selenium автоматизирует работу браузеров, которые загружают веб-сайт, извлекают необходимые данные и даже делают скриншоты или подтверждения (asserts) тех или иных действий на сайте.

У Selenium нет своего собственного веб-браузера, для запуска он использует браузеры сторонних производителей. Например, если вы запустите Selenium Firefox, то в буквальном смысле сможете увидеть, как на вашем экране откроется Firefox, походить по сайту и выполнить действия, указанные в программном коде. Хотя наблюдать за выполнением скриптов интересно, я все же предпочитаю тихо запускать скрипты в фоновом режиме, поэтому использую инструмент под названием PhantomJS вместо фактического браузера.

PhantomJS известен как «браузер без головы». Он загружает веб-сайты в память и выполняет JavaScript на странице, обходясь без графического интерфейса. Объединив возможности Selenium и PhantomJS, вы получите довольно мощный веб-скрапер, который с легкостью обрабатывает cookies, JavaScript, заголовки и все остальное, что вам нужно.

Вы можете скачать библиотеку Selenium с официального сайта или использовать сторонние инсталляторы (например, pip), чтобы установить ее из командной строки.

PhantomJS можно скачать с официального сайта. Поскольку PhantomJS – это полнофункциональный браузер (хоть и без графического интерфейса), который не является библиотекой Python, его нужно сначала загрузить и установить, при этом его нельзя установить с помощью pip.

Хотя в Интернете довольно много страниц, которые используют Ajax для загрузки данных (в частности, Google), я создала тестовую страницу на *http://bit.ly/1HYuH0L*, чтобы проверить работу наших скраперов. Эта страница содержит определенный текст, закодированный в HTML, который после двухсекундной задержки замененяется Ajax-генерируемым контентом. Если бы мы собирали данные с этой страницы, используя традиционные методы, мы извлекли бы страницу загрузки, фактически не получив интересующей нас информации.

Библиотека Selenium представляет собой API под названием *WebDriver*. WebDriver чем-то напоминает браузер, который может загружать сайты, но при этом его еще можно использовать как объект BeautifulSoup для поиска элементов страницы, взаимодействия с элементами страницы (отправить текст, щелкнуть и т. д.), а также выполнения других действий для продолжения работы веб-скрапера.

Код, приведенный ниже, извлекает текст, скрытый за Ajax-стеной на тестовой странице:

```
from selenium import webdriver
import time

driver = webdriver.PhantomJS(executable_path='')
driver.get("http://pythonscraping.com/pages/javascript/ajaxDemo.html")
time.sleep(3)
print(driver.find_element_by_id("content").text)
driver.close()
```

Селекторы Selenium

В предыдущих главах мы выбирали элементы HTML-страницы, используя селекторы BeautifulSoup, например find и findAll. Selenium использует совершенно новый набор селекторов для поиска элемента в DOM-дереве WebDriver, несмотря на то что у них довольно простые имена.

В данном примере мы использовали селектор `find_element_by_id`, хотя другие селекторы, приведенные ниже, тоже сработали бы:

```
driver.find_element_by_css_selector("#content")
driver.find_element_by_tag_name("div")
```

Конечно, если вы хотите выбрать несколько элементов на странице, большинство этих селекторов просто возвратит питоновский список элементов с помощью `elements` (т. е. употребляем название селектора во множественном числе):

```
driver.find_elements_by_css_selector("#content")
driver.find_elements_by_css_selector("div")
```

Разумеется, если вы по-прежнему хотите использовать BeautifulSoup для парсинга контента, делайте это с помощью функции WebDriver `page_source`, которая возвращает исходный код страницы (просматриваемый в момент вызова с помощью DOM) в виде строки:

```
pageSource = driver.page_source
bsObj = BeautifulSoup(pageSource)
print(bsObj.find(id="content").get_text())
```

Этот код создает новый Selenium WebDriver с помощью библиотеки PhantomJS, который сообщает WebDriver о том, что нужно загрузить страницу и затем выдержать трехсекундную паузу, перед тем как начать поиск страницы, которая содержит (надеюсь, уже загруженный) контент.

В зависимости от места установки PhantomJS вам, возможно, потребуется явно указать Selenium правильный путь при создании нового PhantomJS WebDriver:

```
driver = webdriver.PhantomJS(executable_path='/path/to/download/
                                phantomjs-1.9.8-macosx/bin/phantomjs')
```

Если все сконфигурировано правильно, то выполнение скрипта займет несколько секунд, и в итоге будет получен следующий текст:

```
Here is some important text you want to retrieve!
A button to click!
```

Отметим, что хотя сама страница содержит кнопку HTML, фунция `.text` извлекает текстовое значение кнопки таким же образом, каким она извлекает весь остальной контент страницы.

Если значение аргумента функции `time.sleep` (задает паузу в работе скрипта) изменить с 3 на 1, будет возвращен первоначальный текст:

```
This is some content that will appear on the page while it's loading.
 You don't care about scraping this.
```

Несмотря на то что это решение работает, оно в некоторой степени неэффективно, и его реализация может вызвать проблемы, если речь идет о масштабном проекте. Время загрузки страниц постоянно меняется в зависимости от нагрузки на сервер, а также меняется скорость соединения. Хотя загрузка этой страницы должна занимать чуть больше двух секунд, мы даем ей целых три секунды, чтобы убедиться, что она загрузилась полностью. Более эффективное решение заключается в том, чтобы периодически проверять наличие некоторого эле-

мента на полностью загруженной странице и возвращать результат только тогда, когда этот элемент присутствует.

Код, приведенный ниже, перед тем как объявить страницу полностью загруженной, проверяет наличие кнопки с идентификатором `loadedButton`: `from selenium import webdriver`.

```
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC

driver = webdriver.PhantomJS(executable_path='')
driver.get("http://pythonscraping.com/pages/javascript/ajaxDemo.html")
try:
    element = WebDriverWait(driver, 10).until(
                    EC.presence_of_element_located((By.ID, "loadedButton")))
finally:
    print(driver.find_element_by_id("content").text)
    driver.close()
```

В этом коде мы импортировали из Selenium Webdriver `WebDriverWait` и `expected_conditions`, которые здесь используются для реализации *неявного ожидания* (*implicit wait*).

Неявное ожидание отличается от явного тем, что оно указывает Webdriver опрашивать DOM в течение конкретного периода времени при поиске элементов, в случае если они не появились сразу же, а явное ожидание задает жесткий временной интервал, как в предыдущем примере, в котором ожидание составляет три секунды. В неявном ожидании отслеживание готовности DOM задается `expected_condition` (обратите внимание, здесь мы для краткости использовали разрешенное сокращение `EC`). В библиотеке Selenium ожидаемыми условиями могут быть различные события, среди них:

- всплывание окна предупреждения;
- выделение элемента страницы (например, текстового поля);
- изменение заголовка страницы или вывод текста на странице или в определенном элементе;
- появление или исчезновение элемента из DOM.

Конечно, большая часть этих ожидаемых условий требует, чтобы вы указали элемент для наблюдения. Для поиска элементов в Selenium используются *локаторы (locators)*. Обратите внимание, что локаторы – это не то же самое, что селекторы (смотрите «Селекторы Selenium» для получения дополнительной информации о селекторах). Локатор – это строка, уникально идентифицирующая элемент страницы. Объект `By` задает критерий поиска.

В коде, приведенном ниже, локатор используется для поиска элементов по id `loadedButton`:

```
EC.presence_of_element_located((By.ID, "loadedButton"))
```

Кроме того, локаторы можно использовать для создания селекторов, используя функцию WebDriver `find_element`:

```
print(driver.find_element(By.ID, "content").text)
```

который, разумеется, с функциональной точки зрения равнозначен строке кода:

```
print(driver.find_element_by_id("content").text)
```

Если вы не хотите использовать локатор, не используйте, это избавит вас от необходимости импорта. Однако это очень удобный инструмент, который используется для различных задач и обладает значительной гибкостью.

Ниже приводятся способы использования локаторов для поиска элементов с помощью объекта `By`:

`ID`
: Используется в примере, ищет элементы по атрибуту `ID`.

`CLASS_NAME`
: Используется для поиска элементов по атрибуту `class`. Почему эта функция называется `CLASS_NAME`, а не просто `CLASS`? В библиотеке Selenium для языка Java использование формы `Object.class` проблематично, поскольку здесь `.class` является зарезервированным методом. Чтобы синтаксис Selenium был универсален для различных языков, вместо `CLASS` решили использовать `CLASS_NAME`.

`CSS_SELECTOR`
: Ищет элементы по названию `class`, `id` и `tag`, используя соглашение `#idName`, `.className`, `tagName`.

`LINK_TEXT`
: Ищет ссылки с указанным текстом. Например, ссылку «Next» можно найти с помощью `(By.LINK_TEXT, "Next")`.

`PARTIAL_LINK_TEXT`
: Аналогичен `LINK_TEXT`, но ищет ссылки по части с указанным текстом.

`NAME`
: Ищет HTML-теги по атрибуту `name`. Это удобно для HTML-форм.

`TAG_NAME`
Ищет HTML-теги по имени тега.

`XPATH`
Использует `XPath`-выражение (синтаксис, который будет описан ниже в «Синтаксис XPATH») для поиска элементов.

Синтаксис XPath

XPath (сокращение от XML Path) – это язык запросов, используемый для перемещения по XML-документу и выбора его элементов. Он был разработан W3C в 1999 году и иногда используется в таких языках, как Python, Java и C#, при работе с XML-документами.
В отличие от BeautifulSoup, многие библиотеки, использованные в этой книге, поддерживает XPath. Этот язык часто используется в тех же целях, что и CSS-селекторы (например, `mytag#idname`), хотя он предназначен для работы с более обобщенными XML-документами, а не с HTML-документами, в частности.
В синтаксисе XPath есть четыре основных понятия.

- *Корневые узлы и некорневые узлы:*
  - `/div` выберет узел div, только если он находится в корне документа;
  - `//div` выбирает все дивы в любом месте документа.
- *Выбор атрибутов:*
  - `//@href` выбирает все узлы с атрибутом `href`;
  - `//a[@href='http://google.com']` выбирает все ссылки в документе, которые указывают на Google.
- *Выбор узлов по их расположению:*
  - `//a[3]` выбирает третью ссылку в документе;
  - `//table[last()]` выбирает последнюю таблицу в документе;
  - `//a[position() < 3]` выбирает первые три ссылки в документе.
- *Звездочки (*) соответствуют любому набору символов или узлов и могут быть использованы в различных ситуациях:*
  - `//table/tr/*` выбирает все дочерние элементы тегов `tr` во всех таблицах (это хорошо подходит для отбора ячеек, использующих как теги th, так и теги td);
  - `//div[@*]` выбирает все теги `div` с любыми атрибутами.

Конечно, у синтаксиса XPath есть множество дополнительных возможностей. За эти годы он превратился в сложный язык запросов, использующий булеву логику, функции (например, `position()`), а также ряд операторов, которые здесь не рассматриваются.
Если у вас есть задачи отбора элементов в HTML- или XML-документах, которые нельзя решить с помощью функций, рассмотренных здесь, обратитесь к странице синтаксиса Microsoft XPath.

Обработка редиректов

Клиентские редиректы – это перенаправления страниц, которые выполняются в браузере с помощью JavaScript до отправки содержимого страницы, в отличие от серверных редиректов. Иногда, зайдя на

страницу в браузере, бывает сложно понять разницу между ними. Перенаправление может произойти так быстро, что вы не заметите каких-либо задержек загрузки и подумаете, что клиентский редирект является на самом деле серверным редиректом.

Однако при проведении веб-скрапинга разница становится очевидной. Серверный редирект в зависимости от способа его обработки можно легко отследить с помощью питоновской библиотеки urllib, не прибегая к помощи Selenium (для получения дополнительной информации обратитесь к главе 3). Клиентские редиректы вообще не будут обработаны, если не выполнен JavaScript.

Selenium обрабатывает эти JavaScript-редиректы таким же образом, каким он выполняет любой JavaScript. Однако основная проблема, возникающая при работе с этими редиректами, заключается в том, чтобы определить, когда нужно остановить обработку страницы, т. е. как определить момент, когда перенаправление выполнено. Тестовая страница на *http://bit.ly/1SOGCBn* – пример такого редиректа с двухсекундной паузой.

Мы можем определить этот редирект, «отследив» определенный элемент в DOM при первоначальной загрузке страницы и затем повторно вызывая его, пока Selenium не выдаст исключения StaleElementReferenceException, т. е. элемент больше не привязан к DOM-структуре страницы и перенаправление выполнено:

```
from selenium import webdriver
import time
from selenium.webdriver.remote.webelement import WebElement
from selenium.common.exceptions import StaleElementReferenceException

def waitForLoad(driver):
    elem = driver.find_element_by_tag_name("html")
    count = 0
    while True:
        count += 1
        if count > 20:
            print("Timing out after 10 seconds and returning")
            return
        time.sleep(.5)
        try:
            elem == driver.find_element_by_tag_name("html")
        except StaleElementReferenceException:
            return

driver = webdriver.PhantomJS(executable_path='<Path to Phantom JS>')
driver.get("http://pythonscraping.com/pages/javascript/redirectDemo1.html")
```

```
waitForLoad(driver)
print(driver.page_source)
```

Этот скрипт обращается к странице каждые 0,5 секунды и ищет тег `html`. Время ожидания – 10 секунд (20 попыток по 0,5 секунды). При этом паузу между обращениями и количество обращений к странице можно легко увеличить или уменьшить по мере необходимости.

Глава 11

Обработка изображений и распознавание текста

Машинное зрение, начиная от разработки беспилотных автомобилей Google и заканчивая автоматами, распознающими фальшивые деньги, – это огромное поле деятельности с амбициозными целями и перспективными результатами. В этой главе речь пойдет лишь об одном очень маленьком аспекте машинного зрения: о распознавании текста, в частности о том, как распознавать и работать с текстовыми изображениями, найденными в Интернете, используя различные библиотеки Python.

Использование изображения вместо текста – распространенный метод, применяющийся в тех случаях, когда вы не хотите, чтобы текст был найден и прочитан ботами. Его часто можно встретить при заполнении контактной формы, когда адрес электронной почты частично или полностью представлен в виде изображения. В зависимости от того, насколько хорошо этот метод реализован, он, возможно, даже покажется человеку непримечательным, однако ботам будет очень тяжело прочитать эти изображения и этой меры достаточно, чтобы остановить бóльшую часть спамеров.

Разумеется, тест CAPTCHA использует в своих интересах тот факт, что пользователи, в отличие от ботов, без труда могут прочитать изображения безопасности. Некоторые тесты CAPTCHA труднее, чем остальные, и эту проблему мы будем решать позже.

Но CAPTCHA – это не единственное место в Интернете, где скраперам нужно помочь преобразовать изображение в текст. Даже в наше

время еще многие документы просто отсканированы с бумажных носителей и размещены в Интернете, что делает их недоступными для работы, несмотря на то что они «спрятаны на виду у всех». Если нет возможности преобразовать изображение в текст, единственный способ сделать эти документы доступными для работы – набрать их вручную, но ни у кого на это нет времени.

Перевод изображения в текст называется *оптическим распознаванием символов (optical character recognition или OCR)*. Есть несколько больших библиотек, которые способны выполнить OCR, а также много других библиотек, которые поддерживают их или построены на их основе. Иногда эта система библиотек может быть довольно сложной, поэтому я рекомендую вам прочитать следующий раздел, прежде чем перейти к упражнениям этой главы.

Обзор библиотек

Python – великолепный язык, предназначенный для обработки и чтения изображений, распознавания изображений на основе машинного обучения и даже создания изображений. Хотя существует большое количество библиотек, которые можно использовать для обработки и чтения изображений, мы сосредоточимся на двух: Pillow и Tesseract.

Обе библиотеки можно установить, скачав с официальных вебсайтов и установив из исходников (*http://bit.ly/1FVNpnq* и *http://bit.ly/1Fnm6yt*), либо воспользоваться сторонними инсталляторами (например, pip), введя ключевые слова «pillow» и «pytesseract» соответственно.

Pillow

Хотя Pillow не является полномасштабной библиотекой обработки изображений, в ней есть всё, что нужно, с запасом, если только вы не занимаетесь наукой, переписываете Photoshop или делаете что-то подобное. Кроме того, она имеет хорошо задокументированный код, который чрезвычайно прост в использовании.

Библиотека Pillow, являющаяся форком Python Imaging Library (PIL) для Python 2.x, дополнительно поддерживает Python 3.x. Как и ее предшественница, Pillow позволяет легко импортировать и обрабатывать изображения с помощью различных фильтров, масок и даже пиксельных преобразований:

```
from PIL import Image, ImageFilter

kitten = Image.open("kitten.jpg")
blurryKitten = kitten.filter(ImageFilter.GaussianBlur)
blurryKitten.save("kitten_blurred.jpg")
blurryKitten.show()
```

В примере выше изображение *kitten.jpg* откроется в вашем просмотрщике изображений по умолчанию, к нему будет применено размытие, и оно будет сохранено как *kitten_blurred.jpg* в том же самом каталоге.

Мы будем использовать Pillow для выполнения предварительной обработки изображений, чтобы сделать их более машиночитаемыми, однако, как упоминалось ранее, помимо этих простых манипуляций, существует еще много возможностей, которые можно реализовать с помощью этой библиотеки. Для получения дополнительной информации ознакомьтесь с документацией к библиотеке Pillow.

Tesseract

Tesseract – это OCR-библиотека. Спонсоруемая Google (компанией, которая известна своими технологиями OCR и машинного обучения), Tesseract общепризнанно считается наилучшей и наиболее точной OCR-системой с открытым исходным кодом.

Кроме того что Tesseract дает высокую точность, она является чрезвычайно гибкой. Ее можно научить распознавать любое количество шрифтов (при условии, что символы шрифта используют одни и те же стандартные для данного шрифта характеристики), и данный факт можно использовать для распознавания любого символа Unicode.

В отличие от библиотек, которыми мы до сих пор пользовались в этой книге, Tesseract – это инструмент на основе командной строки, написанный на Python, а не библиотека, запускаемая с помощью оператора `import`. После установки ее нужно запустить с помощью команды `tesseract` за пределами Python.

Установка Tesseract

Для пользователей Windows существует удобный инсталлятор. На момент написания книги использовалась версия 3.02, хотя новые версии тоже должны прекрасно подойти.

Пользователи Linux могут установить Tesseract с помощью `apt-get`:

```
$sudo apt-get tesseract-ocr
```

Установка Tesseract на Mac немного сложнее, хотя ее довольно легко установить с помощью различных сторонних инсталляторов, например с помощью Homebrew, который использовался в главе 5 для установки MySQL. Например, вы можете установить Homebrew и использовать его для инсталляции Tesseract, написав две строки кода:

```
$ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/ \
          install/master/install)"
$brew install tesseract
```

Кроме того, Tesseract можно установить из исходника, размещенного на странице загрузки проекта.

Чтобы использовать некоторые возможности Tesseract, например обучить программу распознавать новые символы (об этом речь пойдет позже), вам дополнительно потребуется задать новую переменную окружения `$TESSDATA_PREFIX`, чтобы указать место хранения файлов данных.

В большинстве операционных систем Linux и Mac OS X вы можете сделать это с помощью:

```
$export TESSDATA_PREFIX=/usr/local/share/
```

Обратите внимание, что для Tesseract `/usr/local/share/` – это месторасположение данных по умолчанию, хотя при установке Tesseract вам следует убедиться в этом.

Аналогично в Windows вы можете использовать следующую команду, чтобы задать переменную среды:

```
$pip install numpy
```

NumPy

Хотя NumPy не требуется для непосредственного оптического распознавания символов, вам она понадобится, если вы захотите обучить Tesseract распознаванию дополнительных наборов символов или шрифтов, о которых мы поговорим в этой главе позже. NumPy – очень мощная библиотека, использующая линейную алгебру и другие распространенные математические алгоритмы. NumPy хорошо дополняет Tesseract, потому что она позволяет представить изображения в математическом виде и обрабатывать их как большие массивы пикселей.

Как всегда, NumPy можно установить с помощью любого стороннего инсталлятора, например с помощью pip:

```
$pip install numpy
```

Обработка хорошо отформатированного текста

Если повезет, то обрабатываемый текст будет преимущественно чистым и хорошо отформатированным. Как правило, хорошо отформатированный текст отвечает ряду требований, хотя граница между «грязным» и «хорошо отформатированным» текстом может быть условной.

В целом хорошо форматированный текст:

- написан одним стандартным шрифтом (за исключением рукописных, курсивных или чрезмерно «декоративных» шрифтов);
- скопированный или сфотографированный текст имеет максимально четкие линии без каких-либо артефактов или темных пятен;
- хорошо выровнен, без наклонных букв;
- не пропадает на изображении, не выходит за пределы изображения и не располагается по краям изображения.

Некоторые недостатки текста можно исправить при первичной обработке. Например, изображения можно преобразовать в оттенки серого, можно отрегулировать яркость и контрастность, или изображение можно обрезать и повернуть по мере необходимости. Однако существуют некоторые фундаментальные ограничения, которые могут потребовать более тщательного обучения. Смотрите «Чтение CAPTCHA и обучение Tesseract».

Рисунок 11.1 – это идеальный пример хорошо форматированного текста.

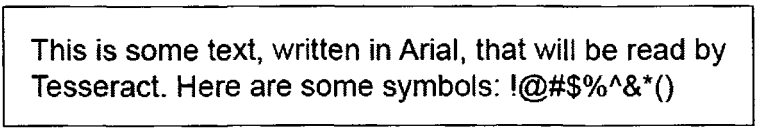


Рис. 11.1 ❖ Текст, сохраненный как файл .tiff и предназначенный для распознавания Tesseract

Вы можете запустить Tesseract из командной строки, чтобы прочитать этот файл и записать результаты в виде текстового файла:

```
$tesseract text.tif textoutput | cat textoutput.txt
```

Вывод – строка с информацией о том, что библиотека Tesseract запущена, затем приводится содержимое вновь созданного файла *textoutput.txt*:

```
Tesseract Open Source OCR Engine v3.02.02 with Leptonica
This is some text, written in Arial, that will be read by
Tesseract. Here are some symbols: !@#$%"&'()
```

Вы можете увидеть, что результаты распознавания довольно точные, хотя символы «^» и «*» были интерпретированы как двойная кавычка и одинарная кавычка, соответственно. В целом читать текст довольно комфортно.

После размытия текстового изображения, внесения некоторых артефактов сжатия JPG и добавления небольшого фонового градиента результаты становятся гораздо хуже (рис. 11.2).

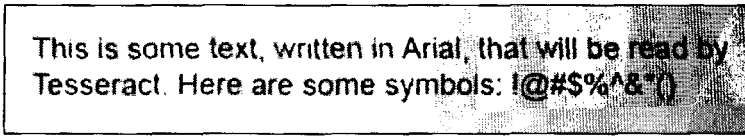


Рис. 11.2 ❖ К сожалению, многие документы, с которыми вы столкнетесь в Интернете, будет выглядеть именно так, а не как в предыдущем примере

Tesseract не может обработать это изображение так же хорошо, как предыдущее, главным образом из-за фонового градиента, и выдает следующий результат:

```
This is some text, written In Arlal, that"
Tesseract. Here are some symbols: _
```

Обратите внимание, что текст обрезается, как только фоновый градиент делает текст максимально нечитаемым и последний символ каждой строки является неправильным, поскольку Tesseract безуспешно пытается придать тексту смысл. Кроме того, JPG-артефакты и размытие не позволяют Tesseract отличить строчную «i» от заглавной «I» и цифры «1».

Это как раз та ситуация, когда использование скрипта Python для очистки изображений приходится как нельзя кстати. С помощью библиотеки Pillow мы можем создать пороговый фильтр, чтобы избавиться от серого фона, повысить четкость текста и сделать изображение более понятным для Tesseract:

```
from PIL import Image
import subprocess

def cleanFile(filePath, newFilePath):
    image = Image.open(filePath)
```

```
    #Задаем пороговое значение для данного изображения и сохраняем
    image = image.point(lambda x: 0 if x<143 else 255)
    image.save(newFilePath)

    #Вызываем tesseract, чтобы выполнить OCR вновь созданного
    #изображения
    subprocess.call(["tesseract", newFilePath, "output"])

    #Открываем и читаем полученный в результате файл данных
    outputFile = open("output.txt", 'r')
    print(outputFile.read())
    outputFile.close()

cleanFile("text_2.png", "text_2_clean.png")
```

Получившееся изображение, автоматически созданное как *text_2_clean.png*, показано на рис. 11.3:

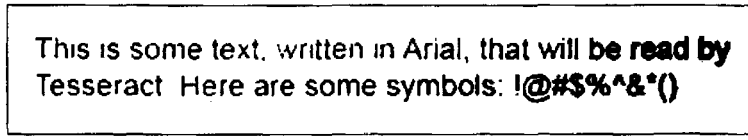


Рис. 11.3 ❖ Это изображение было получено путем обработки предыдущей «грязной» версии изображения с помощью порогового фильтра

Помимо некоторых сдва заметных или пропущенных знаков пунктуации, текст читаем, по крайней мере, для нас. Tesseract выдает свой наилучший результат:

```
This us some text' written In Anal, that will be read by
Tesseract Here are some symbols: !@#$%"&'()
```

Точки и запятые, будучи чрезвычайно маленькими, становятся первыми жертвами наших манипуляций с изображением и практически не видны, как с нашей точки зрения, так и с точки зрения Tesseract. Кроме того, слово «Arial» неправильно распознано как «Anal», буквы «r» и «i» были распознаны Tesseract как одна буква «n».

Однако это улучшение, по сравнению с предыдущей версией, в которой почти половина текста была обрезана.

Похоже, что самое слабое место Tesseract – это обработка фоновых рисунков с непостоянной яркостью. Алгоритмы Tesseract пытаются автоматически задать контрастность изображения перед чтением текста, но вы, вероятно, сможете получить лучшие результаты, само-

стоятельно настроив контрастность с помощью такого инструмента, как библиотека Pillow.

Изображения, которые перевернуты, практически не содержат текста или имеют другие проблемы, необходимо предварительно обработать, перед тем как отправлять в Tesseract.

Скрапинг текста с изображений, размещенных на веб-сайтах

Использование Tesseract для чтения текста из изображения, размещенного на вашем жестком диске, возможно, не покажется вам чем-то захватывающим, но может стать очень мощным инструментом в ходе веб-скрапинга. Изображения могут непреднамеренно усложнить распознавание текста на веб-сайтах (например, меню на сайте ресторана, выполненное в виде картинок с причудливым текстом), кроме того, они могут целенаправленно скрывать текст, как я покажу в следующем примере.

Хотя файл *robots.txt* сайта Amazon позволяет выполнять скрапинг страниц с товарами, боты не собирают превью книг. Это обусловлено тем, что превью книг загружаются с помощью Ajax-скрипта, активируемого пользователем, а сами изображения надежно скрыты под слоями из тегов `<div>`. На самом деле обычному посетителю сайта они скорее покажутся флеш-презентацией, чем файлами изображений. Конечно, даже если бы мы и получили доступ к изображениям, еще остается немаловажный вопрос, как их преобразовать в текст.

Скрипт, приведенный ниже, тихо совершает этот подвиг: он переходит к изданию «Войны и мира» Льва Толстого, напечатанному крупным шрифтом[1], открывает просмотрщик, собирает URL-адреса изображений, а затем последовательно скачивает, читает и выводит текст каждого изображения. Поскольку это сложный код, использующий несколько принципов, о которых рассказывалось в предыдущих главах, я добавила везде комментарии, чтобы вам было легче понять его:

```
import time
from urllib.request import urlretrieve
```

[1] Когда дело доходит до обработки текста, которому Tesseract не был обучен, программа гораздо лучше распознает широкоформатные издания книг, особенно если иллюстрации в них маленького размера. В следующем разделе мы расскажем, как обучить Tesseract различным шрифтам, что позволит ему читать гораздо меньшие размеры шрифтов, в том числе превью для книжных изданий небольшого формата!

```
import subprocess
from selenium import webdriver

#Создаем новый драйвер Selenium
driver = webdriver.PhantomJS(executable_path='<Path to Phantom JS>')
driver.get(
      "http://www.amazon.com/War-Peace-Leo-Nikolayevich-Tolstoy/dp/1427030200")
time.sleep(2)

#Щелкаем по кнопке предпросмотра книги
driver.find_element_by_id("sitbLogoImg").click()
imageList = set()

#Ждем, пока страница загрузится
time.sleep(5)
#Пока кнопка со стрелкой вправо кликабельна, перелистываем страницу
while "pointer" in driver.find_element_by_id("sitbReaderRightPageTurner")
                                              .get_attribute("style"):
    driver.find_element_by_id("sitbReaderRightPageTurner").click()
    time.sleep(2)
    #Ищем URL-адреса изображения на всех загруженных страницах (мы уже
        #загрузились сразу несколько страниц, но дубликаты не будут
        #добавлены в набор)
    pages = driver.find_elements_by_xpath("//div[@class='pageImage']/div/img")
    for page in pages:
        image = page.get_attribute("src")
        imageList.add(image)

driver.quit()

#Начинаем обработку изображений, собранных с URL-адресов, с помощью
#Tesseract
for image in sorted(imageList):
    urlretrieve(image, "page.jpg")
    p = subprocess.Popen(["tesseract", "page.jpg", "page"],
                         stdout=subprocess.PIPE,stderr=subprocess.PIPE)
    p.wait()
    f = open("page.txt", "r")
    print(f.read())
```

Tesseract идеально распознает текст книги, как это видно на странице предварительного просмотра 6:

```
6

"A word of friendly advice, mon
cher. Be off as soon as you can,
that's all I have to tell you. Happy
he who has ears to hear. Good-by,
my dear fellow. Oh, by the by!" he
```

```
shouted through the doorway after
Pierre, "is it true that the countess
has fallen into the clutches of the
holy fathers of the Society of je-
sus?"

Pierre did not answer and left Ros-
topchin's room more sullen and an-
gry than he had ever before shown
himself.
```

Однако когда текст напечатан на цветном фоне передней и задней обложки, он становится непонятным:

```
WEI' nrrd Peace
Len Nlkelayevldu Iolfluy

Readmg shmdd be ax
wlnvame asnossxble Wenfler
an mm m our cram: Llhvary

- Leo Tmsloy was a Russian rwovelwst
I and moval phflmopher med lur
A ms Ideas 01 nonviolenx reswslance m 5 We range 0, "and"
```

Конечно, вы можете использовать библиотеку Pillow для выборочной очистки изображений, но это может быть крайне трудоемким занятием для процесса, который предполагает по возможности минимальное участие человека.

В следующем разделе рассматривается другой подход к решению проблемы нечитаемого текста, особенно если вы не против того, чтобы потратить немного времени на обучение Tesseract. «Познакомив» Tesseract с большой коллекцией текстовых изображений, которые уже прочитаны, вы научите Tesseract распознавать в будущем данный шрифт с гораздо большей точностью и достоверностью, даже несмотря на случайный фон и расположение на странице.

Чтение CAPTCHA и обучение Tesseract

Хотя слово «CAPTCHA» (капча) знакомо большинству людей, очень немногие знают, что оно означает: Completely Automated Public Turing test to tell Computers and Humans Apart (полностью автоматизированный публичный тест Тьюринга для различения компьютеров и людей). Этот неуклюжий акроним намекает на то, что капча создает людям и роботам препятствия, чтобы предотвратить нежелательное использование прекрасно работающих веб-интерфейсов.

Тест Тьюринга был впервые описан Аланом Тьюрингом в его статье 1950 года «Computing Machinery and Intelligence» («Вычислительные машины и разум»). В этой статье он описал ситуацию, когда человек может общаться как с людьми, так и с программами искусственного интеллекта через компьютерный терминал. Если во время случайного разговора человек не смог отличить человека от программы искусственного интеллекта, значит, программа прошла тест Тьюринга, и искусственный интеллект, по мнению Тьюринга, действительно умеет «обдумывать» свои цели и намерения.

Забавно, что за последних 60 лет мы прошли путь от использования этих тестов для проверки машин к использованию их для проверки самих себя, получив смешанные результаты. Как известно, сложная система защиты reCAPTCHA, приобретенная Google и в настоящее время являющаяся наиболее популярной среди сайтов, заботящихся о безопасности, блокирует около 25% настоящих пользователей[1].

Большинство других капчей несколько проще. Например, у Drupal, широко известной системы управления контентом на основе PHP, есть популярный модуль CAPTCHA, который может генерировать изображения капчи различной сложности. Изображение по умолчанию выглядит как на рис. 11.4.

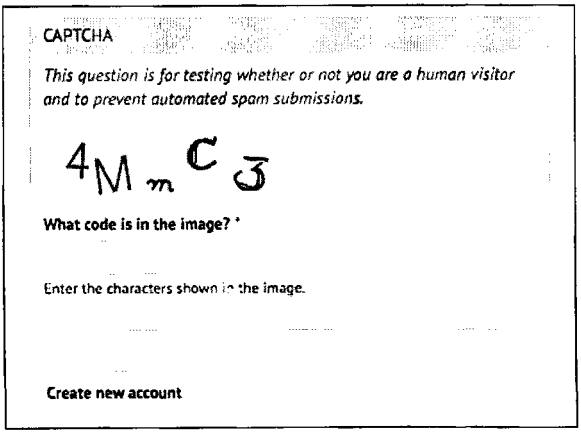


Рис. 11.4 ❖ Пример текста капчи по умолчанию для модуля CAPTCHA в Drupal

[1] См. *http://bit.ly/1HGTbGf*.

Почему эта капча легко читается как людьми, так и машинами, по сравнению с другими капчами?

- Символы не накладываются друг на друга, не пересекаются друг с другом по горизонтали. То есть вокруг каждого символа можно нарисовать аккуратный прямоугольник, не затрагивая любого другого символа.
- Здесь нет фоновых изображений, линий и другого отвлекающего мусора, который может смутить OCR-программу.
- На этом изображении не видно, но шрифт капчи не слишком вариативен. Здесь используется сочетание шрифта без засечек (взгляните на символы «4» и «M») и рукописного шрифта (взгляните на символы «m», «C» и «3»).
- Здесь мы видим высокую контрастность между белым фоном и символами, окрашенными в темные цвета.

Однако эта капча использует несколько уловок, которые затрудняют ее распознавание OCR-программой:

- используются как буквы, так и цифры, что увеличивает количество потенциальных символов;
- случайный характер наклона букв может запутать OCR-программу, но легко читается человеком;
- странный рукописный шрифт представляет особую трудность наряду с вертикальными линиями в символах «C» и «3» и необычно маленькой строчной буквой «m», для их правильного распознавания потребуется дополнительное обучение компьютерной программы.

Когда мы запускаем Tesseract для обработки этого изображения, используя команду:

```
$tesseract captchaExample.png output
```

мы получаем файл *output.txt*:

```
4N\,,,C<3
```

Программа правильно распознает 4, C и 3, но очевидно, что полностью распознать капчу она в ближайшее время не сможет.

Обучение Tesseract

Чтобы научить Tesseract распознавать текст, будь то непонятный и трудночитаемый шрифт или капча, нужно дать ему несколько вариантов написания каждого символа.

Это та часть книги, когда можно сходить выпить чаю, потому что предстоит пара часов довольно скучной работы. Первый шаг заклю-

чается в том, чтобы скачать большое количество примеров вашей капчи в отдельный каталог. Количество примеров, используемых для обучения, будет зависеть от сложности капчи. Для обучения я использовала 100 файлов-примеров (всего 500 символов, или около 8 примеров на один символ в среднем), и мне кажется, что этого будет достаточно.

Совет: я рекомендую давать имена изображениям в соответствии с капчей, которую они содержат (т. е. *4MmC3.jpg*). Я пришла к выводу, что это помогает быстро находить ошибки при одновременной работе с большим количеством файлов, – можно просматривать все файлы в виде эскизов и довольно удобно сравнивать изображение с его именем. Это значительно упрощает поиск ошибок на последующих этапах.

Второй шаг заключается в том, чтобы сообщить Tesseract, что представляет из себя каждый символ, и указать, где он находится в изображении. Это подразумевает создание box-файлов, по одному для каждого изображения капчи. Box-файл выглядит следующим образом:

```
4 15 26 33 55 0
M 38 13 67 45 0
m 79 15 101 26 0
C 111 33 136 60 0
3 147 17 176 45 0
```

Box-файл содержит строки, в каждой строке записывается один символ с координатами прямоугольника вокруг изображения (четыре цифры), а последняя цифра – это «номер страницы», который используется для обучения на основе многостраничных документов (0 в нашем случае).

Очевидно, что эти box-файлы непросто создать вручную, однако есть целый ряд инструментов, которые помогут вам. Мне нравится онлайн-инструмент Tesseract OCR Chopper, потому что он не требует установки или дополнительных библиотек, работает на любой машине, где есть браузер, и относительно прост в использовании: загрузите изображение, нажмите кнопку «Add» в нижней части окна, если вам нужны еще box-файлы, скорректируйте размер файлов при необходимости, скопируйте и вставьте ваш текст в новый файл.

Box-файлы должны быть сохранены в простом текстовом формате с расширением *.box*. Как и файлам изображений, box-файлам лучше давать имена в соответствии с капчей, которая записана в них (например, *4MmC3.box*). Опять же, удобно сравнить содержимое box-файла

с его именем, а затем с соответствующим файлом изображения, отсортировав все файлы в каталоге по имени файла.

И снова вам нужно создать около 100 таких файлов. Кроме того, Tesseract иногда отклоняет файлы как нечитаемые, поэтому у вас должен быть определенный резервный запас. Если вы обнаружили, что результаты распознавания не так хороши, как вы хотелось бы, или Tesseract не может идентифицировать определенные символы, оптимально создать дополнительные файлы для обучения и повторить попытку.

Создав каталог с box-файлами и файлами изображений, скопируйте эти данные в резервную папку, прежде чем проводить какие-либо дальнейшие манипуляции с ними. Хотя в ходе запуска скриптов, выполняющих обучение на основе данных, вряд ли что-то будет удалено, лучше перестраховаться, чем потом сожалеть о времени, которое вы потратили на создание box-файлов. Кроме того, иногда лучше отказаться от захломленного каталога, полного скомпилированных данных, и повторить попытку.

Проведение полного анализа данных и создание файлов для обучения, необходимых для Tesseract, состоят из полдюжины шагов. Есть инструменты, которые могут выполнить это за вас, но, к сожалению, для Tesseract 3.02 нет ни одного такого инструмента.

Я написала решение на языке Python, которое собирает файлы изображений и box-файлы в единый файл и создает все необходимые файлы обучения автоматически.

Основные настройки и шаги, которые выполняет программа, можно посмотреть в ее методах `main` и `runAll`:

```
def main(self):
    languageName = "eng"
    fontName = "captchaFont"
    directory = "<path to images>"

def runAll(self):
    self.createFontFile()
    self.cleanImages()
    self.renameFiles()
    self.extractUnicode()
    self.runShapeClustering()
    self.runMfTraining()
    self.runCnTraining()
    self.createTessData()
```

Единственные три переменные, которые вы должны будете здесь задать, довольно просты:

- `languageName` – трехбуквенный код языка, который позволяет Tesseract понять, с каким языком она работает. В большинстве случаев вы, скорее всего, будете использовать код «eng» для английского языка;
- `FontName` – имя выбранного шрифта. Может быть любым, но с условием, что это должно быть одно слово без пробелов;
- *Каталог, содержащий все файлы изображений и box-файлы*: я рекомендую вам задать абсолютный путь, но если вы используете относительный путь, он должен соответствовать месту, откуда вы запускаете код Python. Если путь является абсолютным, можно запускать код из любого места на вашей машине.

Давайте посмотрим на отдельные функции, использованные в программе.

`createFontFile` создает необходимый файл *font_properties*, который сообщает Tesseract о новом созданном нами шрифте:

```
captchaFont 0 0 0 0 0
```

Этот файл состоит из имени шрифта, после которого следуют единицы и нули, указывающие тип анализируемого шрифта – курсив, жирный и т. д. (обучение шрифтов с помощью этих настроек – интересное упражнение, но, к сожалению, оно выходит за рамки этой книги).

`cleanImages` создает высококонтрастные версии всех найденных файлов изображений, преобразует их в оттенки серого и выполняет другие операции, которые улучшают чтение файлов изображений с использованием OCR-программ. Если вы работаете с изображениями капчи, содержащими визуальный мусор, и его можно легко удалить в ходе последущей обработки, именно с помощью этой функции можно задать эту дополнительную обработку.

`renameFiles` переименовывает все ваши box-файлы и соответствующие файлы изображений в соответствии с требованиями Tesseract (здесь номера файлов – это порядковые номера, необходимые для отдельного хранения файлов):

- *<languageName>.<fontName>.exp<fileNumber>.box*
- *<languageName>.<fontName>.exp<fileNumber>.tiff*

`extractUnicode` анализирует все созданные box-файлы и определяет итоговый набор символов для обучения. Полученный в результате файл Unicode сообщит вам, сколько различных символов найдено, и позволит быстро увидеть пропущенные вами символы.

Следующие три функции: `runShapeClustering`, `runMfTraining` и `runCtTraining` – создают файлы `shapetable`, `pfftable` и `normproto` соответственно. Все они содержат информацию о геометрии и форме каждого символа, а также статистическую информацию, которую Tesseract использует для вычисления вероятности того, что заданный символ относится к тому или иному типу.

Наконец, Tesseract переименовывает каждый каталог со скомпилированными данными так, чтобы соответствующий код языка указывался перед именем каталога (например, *shapetable* переименовывается в *eng.shapetable*), и собирает все эти файлы в итоговый файл обучающих данных eng.traineddata.

Единственное, что вы должны сделать вручную, – это поместить созданный файл *eng.traineddata* в корневую папку *tessdata* с помощью следующих команд:

```
$cp /path/to/data/eng.traineddata $TESSDATA_PREFIX/tessdata
```

Применяя вышеописанные шаги, вы больше не должны испытывать проблем с капчей, распознавать которую мы только что обучили Tesseract. Теперь, когда я хочу с помощью Tesseract прочитать пример изображения, я получаю правильный ответ:

```
$ tesseract captchaExample.png output;cat output.txt
4MmC3
```

Ура! Значительное улучшение, по сравнению с предыдущим результатом «4N\„„C<3».

Это всего лишь краткий обзор возможностей Tesseract по обучению и распознаванию шрифтов. Если вы хотите более глубже изучить вопросы обучения Tesseract, начав собирать свою собственную библиотеку файлов для распознавания капчи или обмениваясь с пользователями по всему миру новыми возможностями распознавания шрифтов, я рекомендую обратиться к документации.

Извлечение CAPTCHA и отправка результатов распознавания

Многие популярные системы управления контентом часто испытывают наплывы спам-регистраций, осуществляемые ботами, которые заранее запрограммированы на определенное месторасположение регистрационной страницы. Например, на *http://pythonscraping.com*

даже капчи (правда, слабой) недостаточно, чтобы притормозить поток регистраций.

Так как же боты делают это? Мы успешно решили задачу распознавания капчей в изображениях, расположенных на нашем жестком диске, но как создать полнофункционального бота? В этом разделе мы свяжем воедино большое количество методов, о которых рассказывалось в предыдущих главах. Если вы еще не знаете о них, то, по крайней мере, бегло просмотрите главы, рассказывающие об отправке форм и загрузке файлов.

Большая часть капчей на основе изображений обладает рядом свойств:

- они являются динамически генерируемыми изображениями, которые создаются серверной программой. У них есть исходное изображение, которое не похоже на обычное изображение, например, `<img src="WebForm.aspx?id=8AP85CQKE9TJ">`, но его можно скачать и работать с ним, как с любым другим изображением;
- решение для данного изображения хранится в серверной базе данных;
- многие капчи имеют свой «срок годности». Как правило, это не является проблемой для ботов, однако выстраивание очереди решений капчи для последующего использования или другие действия, увеличивающие интервал между моментом запроса капчи и моментом отправки решения, могут не увенчаться успехом.

Общий подход в этой ситуации – скачать файл с изображением капчи на жесткий диск, очистить от визуального мусора, применить Tesseract для распознавания изображения и возвратить решение, согласно соответствующему параметру формы.

Я создала страницу на http://pythonscraping.com/humans-only. Она содержит форму комментариев, защищенную капчей. Я хочу продемонстрировать бота, которого надо победить. Бот выглядит следующим образом:

```
from urllib.request import urlretrieve
from urllib.request import urlopen
from bs4 import BeautifulSoup
import subprocess
import requests
from PIL import Image
from PIL import ImageOps

def cleanImage(imagePath):
```

```
    image = Image.open(imagePath)
    image = image.point(lambda x: 0 if x<143 else 255)
    borderImage = ImageOps.expand(image,border=20,fill='white')
    borderImage.save(imagePath)

html = urlopen("http://www.pythonscraping.com/humans-only")
bsObj = BeautifulSoup(html)
#Собираем предварительно заполненные значения полей
imageLocation = bsObj.find("img", {"title": "Image CAPTCHA"})["src"]
formBuildId = bsObj.find("input", {"name":"form_build_id"})["value"]
captchaSid = bsObj.find("input", {"name":"captcha_sid"})["value"]
captchaToken = bsObj.find("input", {"name":"captcha_token"})["value"]

captchaUrl = "http://pythonscraping.com"+imageLocation
urlretrieve(captchaUrl, "captcha.jpg")
cleanImage("captcha.jpg")
p = subprocess.Popen(["tesseract", "captcha.jpg", "captcha"], stdout=
    subprocess.PIPE,stderr=subprocess.PIPE)
p.wait()
f = open("captcha.txt", "r")

#Удаляем любые пробельные символы
captchaResponse = f.read().replace(" ", "").replace("\n", "")
print("Captcha solution attempt: "+captchaResponse)

if len(captchaResponse) == 5:
    params = {"captcha_token":captchaToken, "captcha_sid":captchaSid,
              "form_id":"comment_node_page_form", "form_build_id": formBuildId,
                  "captcha_response":captchaResponse, "name":"Ryan Mitchell",
                  "subject": "I come to seek the Grail",
                  "comment_body[und][0][value]":
                                          "...and I am definitely not a bot"}
    r = requests.post("http://www.pythonscraping.com/comment/reply/10",
                          data=params)
    responseObj = BeautifulSoup(r.text)
    if responseObj.find("div", {"class":"messages"}) is not None:
        print(responseObj.find("div", {"class":"messages"}).get_text())
else:
    print("There was a problem reading the CAPTCHA correctly!")
```

Обратите внимание, есть два условия, вследствие которых этот скрипт может завершится с ошибкой: когда Tesseract не может извлечь все пять символов с картинки (а мы знаем, что все правильные решения этой капчи должны содержать пять символов) или когда он отправляет форму, но капча была решена неправильно. Первая ситуация встречается примерно в 50% случаев, здесь Tesseract не утруждает себя отправкой формы и выдает сообщение об ошибке.

Вторая ситуация происходит примерно в 20% случаев, когда общая точность распознавания капчи составляет 30% (если капча состоит из 5 символов и точность распознавания каждого символа равна 80%, вероятность правильного распознавания всей капчи составляет 0,80*0,80*0,80*0,80*0,80 ~ 0,30, или 30%).

Хотя работа этого скрипта может показаться низкорезультатной, имейте в виду, что, как правило, количество попыток ввода капчи не ограничено, и большую часть этих неудачных попыток можно прервать, не отправляя форму. Если форма отправляется, то, как правило, капча является точной. Если это не убедило вас, имейте в виду, что простое угадывание даст вам уровень точности 0,0000001%. Три четыре запуска программы вместо 900 миллионов угадываний — ощутимая экономия времени!

Глава 12

Обход ловушек в ходе скрапинга

Вы делаете скрапинг сайта, просматриваете результаты и не находите данных, которые так отчетливо отображаются вашим браузером. Или вы отправили форму, которая выглядит безупречно, но веб-сервер выдал отказ. Или вы получили сообщение, что ваш IP-адрес заблокирован сайтом по неизвестным причинам. Однако есть ситуации и похуже.

Существуют баги, которые сложнее всего исправить не только потому, что они возникают неожиданно (скрипт, который прекрасно выполняется на одном сайте, совершенно не работает на практически аналогичном сайте), но и потому, что они не сопровождаются сообщениями об ошибках или трассировками стека, которые можно использовать для отладки скрапера. Вы будете идентифицированы как бот, отклонены и даже не узнаете причину отказа.

В этой книге я уже писала о различных хитростях при работе с вебсайтами (отправка форм, извлечение и очистка сложных данных, выполнение JavaScript и т. д.). Эта глава – своего рода сборная солянка из описанных ранее методов, что обусловлено разнообразием охватываемых в ней тем (HTTP-заголовки, CSS, HTML-формы, и это лишь некоторые из них). Однако все эти методы имеют нечто общее: они предназначены для преодоления препятствий, единственная задача которых – не допустить автоматизированного скрапинга сайта.

Независимо от того, насколько полезной эта информация окажется для вас в данный момент, я настоятельно рекомендую вам, по крайней мере, бегло просмотреть эту главу. Никогда не знаешь, когда именно эта информация поможет вам исправить очень сложную ошибку или вообще предотвратить проблему.

Обратите внимание на этический аспект

В первых нескольких главах этой книги я говорила о «серой правовой зоне» веб-скрапинга, а также о некоторых этических принципах скрапинга. Скажу честно, эта глава с этической точки зрения была, возможно, самой трудной для меня. Мой веб-сайт страдает от наплыва ботов, спамеров, веб-скраперов и всяких нежелательных виртуальных гостей, возможно, что и ваши веб-сайты не избежали этого. Так почему же я рассказываю вам о том, как создать более продвинутых ботов?

Есть несколько причин, почему я считаю эту главу важной:

- есть весомые этические и юридические причины проводить скрапинг некоторых веб-сайтов, которые противятся сбору информации. На предыдущей работе я использовала скрапер для автоматизированного сбора информации с веб-сайтов, которые публиковали в Интернете имена клиентов, адреса, телефонные номера и другую личную информацию без их согласия. Я использовала информацию, собранную скрапером, чтобы официально обратиться к веб-сайтам с требованием удалить эту информацию. Из-за конкуренции эти сайты бдительно охраняют свою информацию от скраперов. Однако моя работа, направленная на обеспечение анонимности моих клиентов (некоторые из них преследовались, были жертвами домашнего насилия, в общем, у них были весомые причины скрывать информацию о себе), была убедительным аргументом для проведения веб-скрапинга, и я была рада, что обладала навыками, необходимыми для проведения этой работы;
- хотя создать сайт, неуязвимый для скраперов, почти невозможно (или, по крайней мере, сайт, на который могут легко зайти лишь законные пользователи), я надеюсь, что информация в этой главе поможет тем, кто хочет защитить свои веб-сайты от вредоносных атак. В этой главе я расскажу о некоторых уязвимостях того или иного метода веб-скрапинга, которые вы можете использовать для защиты Вашего собственного сайта. Имейте в виду, что сегодня большинство ботов в Интернете просто выполняет обширное сканирование информации и уязвимостей, и применение даже пары простых методов, описанных в этой главе, скорее всего, помешает работе 99% скраперов. Однако с каждым месяцем скраперы становятся все изощреннее, и лучше держать ухо востро;

- как и большинство программистов, я просто не верю, что сокрытие какой-либо информации образовательного характера является правильным.

Читая эту главу, имейте в виду, что многие из этих скриптов и описанные методы не нужно тестировать на каждом найденном вами сайте. Это не только не приветствуется, но и может закончиться предупредительным письмом или хуже. Однако я не собираюсь каждый раз предупреждать вас об этом, описывая новый метод. Таким образом, для оставшейся части этой книги актуально высказывание Форреста Гампа: «Это все, что я могу сказать об этом».

Учимся выглядеть как человек

Основная задача сайта, который хочет предотвратить скрапинг, – отличить бота от человека. Хотя большинство техник, используемых сайтами (например, капча), трудно обмануть, есть несколько довольно простых вещей, которые вы можете сделать, чтобы ваш бот выглядел более похожим на человека.

Настройте заголовки

В главе 9 мы использовали модуль `requests` для обработки форм. Модуль `requests` также отлично подходит для настройки заголовков. HTTP-заголовки – это список атрибутов или параметров, отправляемый вами каждый раз, когда вы делаете запрос на веб-сервер. HTTP задает десятки непонятных типов заголовков, большинство из которых обычно не используется. Однако следующие семь полей неизменно используются большинством основных браузеров при установлении соединения (в качестве примера взяты данные моего собственного браузера):

`Host`	https://www.google.com/
`Connection`	keep-alive
`Accept`	text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
`User-Agent`	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_5) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/39.0.2171.95 Safari/537.36
`Referrer`	https://www.google.com/
`Accept-Encoding`	gzip, deflate, sdch
`Accept-Language`	en-US,en;q=0.8

А ниже приводятся заголовки, которые может отправить обычный питоновский скрапер, использующий стандартную библиотеку `urllib`:

`Accept-Encoding`	identity
`User-Agent`	Python-urllib/3.4

Если вы являетесь администратором сайта, пытающимся заблокировать скрапер, какие значения параметров вы пропустите с большей вероятностью?

Установка Requests

Мы устанавливали модуль `requests` в главе 9, но если вы еще этого не сделали, вы можете найти ссылки для скачивания и инструкции по установке на веб-сайте модуля или использовать сторонний инсталлятор.

К счастью, заголовки можно полностью настроить с помощью модуля `requests`. Сайт *https://www.whatismybrowser.com* отлично подходит для тестирования параметров браузера, просматриваемых сервером. Мы выполним скрапинг этого сайта, чтобы проверить настройки обработки cookies, используя следующий скрипт:

```
import requests
from bs4 import BeautifulSoup

session = requests.Session()
headers = {"User-Agent":"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_5)
                        AppleWebKit 537.36 (KHTML, like Gecko) Chrome",
           "Accept":"text/html,application/xhtml+xml,application/xml;
                    q=0.9,image/webp,*/*;q=0.8"}
url = "https://www.whatismybrowser.com/
       developers/what-http-headers-is-my-browser-sending"
req = session.get(url, headers=headers)

bsObj = BeautifulSoup(req.text)
print(bsObj.find("table",{"class":"table-striped"}).get_text)
```

Вывод должен показать, что теперь заголовки идентичны заголовкам, заданным в словаре `headers`.

Хотя веб-сайты могут определить «человечность» с помощью любого параметра HTTP-заголовков, я пришла к выводу, что, как правило, единственным параметром, который действительно имеет значение, является `User-Agent`. Независимо от проекта, над которым вы работаете, лучше установить значение этого параметра на что-то более незаметное, чем `Python-urllib/3.4`. Кроме того, если вы однаж-

ды столкнетесь с очень подозрительным сайтом, заполнение одного из часто используемых, но редко проверяемых заголовков, например `Accept-Language`, может убедить его, что вы являетесь человеком.

Заголовки меняют ваше видение мира

Допустим, вы хотите написать переводчик для исследовательского проекта, но для его тестирования не хватает нужного объема переведенных текстов. Многие крупные сайты делают перевод контента в зависимости от языковых свойств, заданных в заголовках. Простая смена `Accept-Language:en-US` на `Accept-Language:fr` в заголовках позволяет вам получить французскую версию сайта для выполнения перевода (как правило, лучше всего для этих целей подходят сайты крупных международных компаний).

Кроме того, заголовки могут заставить сайт изменить формат представления контента. Например, для мобильных устройств используются урезанные версии сайтов без баннеров, Flash и других отвлекающих факторов. Если вы поменяете значение параметра `User-Agent` на что-то, аналогичное приведенному ниже, вы, возможно, обнаружите, что с этих версий сайтов намного легче собрать информацию!

User-Agent:Mozilla/5.0 (iPhone; CPU iPhone OS 7_1_2 like Mac OS X) AppleWebKit/537.51.2 (KHTML, like Gecko) Version/7.0 Mobile/11D257 Safari/9537.53

Обработка cookies

Правильная обработка cookies может облегчить проблемы веб-скрапинга, хотя cookies могут быть палкой о двух концах. Сайты, которые отслеживают ваши перемещения по сайту с помощью cookies, возможно, попытаются помешать работе скраперов, которые выдают себя своим аномальным поведением, например слишком быстро заполняют формы или посещают слишком много страниц. Несмотря на то что такое поведение можно замаскировать, закрывая и повторно открывая соединения с сайтом или даже меняя IP-адрес (смотрите главу 14 для получения более подробной информации о том, как это сделать), если cookies идентифицируют вас, ваши усилия по маскировке могут быть бесполезны.

Кроме того, cookies могут очень пригодиться для проведения скрапинга. Как показано в главе 9, чтобы войти на сайт, у вас должен быть cookie, который предъявляется каждой странице. Некоторые веб-сайты даже не требуют фактического входа и получения каждый раз нового cookie – достаточно просто предъявить старую копию cookie для «входа» и зайти на сайт.

Если вы выполняете скрапинг отдельного веб-сайта или небольшого количества сайтов, я рекомендую исследовать cookies, сгенерированные этими сайтами, и определить cookies, которые вы хотите

использовать в вашем скрапере. Есть ряд плагинов для браузеров, которые показывают, как создаются cookies, когда вы посещаете сайт и перемещаетесь по нему. Расширение для браузера Chrome EditThis-Cookie является одним из моих любимых.

Обратись к примерам кода в разделе «Работа с логинами и cookies» главы 9 для получения дополнительной информации об обработке cookies с помощью модуля `requests`. Конечно, поскольку модуль `requests` не в состоянии выполнить JavaScript, он не сможет обработать бóльшую часть cookies, сгенерированных современным программным обеспечением для отслеживания, например, Google Analytics. Эти cookies устанавливаются только после выполнения клиентских скриптов (или после выполнения определенных действий на странице, например после щелчка мышью по кнопке во время просмотра страницы). Для их обработки вы должны использовать пакеты Selenium и PhantomJS (мы рассмотрели их установку и основное применение в главе 10).

Вы можете просмотреть cookies, посетив любой сайт (*http://pythonscraping.com* в данном примере) и вызвав `get_cookies()` для webdriver:

```
from selenium import webdriver
driver = webdriver.PhantomJS(executable_path='<Path to Phantom JS>')
driver.get("http://pythonscraping.com")
driver.implicitly_wait(1)
print(driver.get_cookies())
```

Этот код выдает довольно типичный массив cookies, используемых Google Analytics:

```
[{'value': '1', 'httponly': False, 'name': '_gat', 'path': '/', 'expi
ry': 1422806785, 'expires': 'Sun, 01 Feb 2015 16:06:25 GMT', 'secure'
: False, 'domain': '.pythonscraping.com'}, {'value': 'GA1.2.161952506
2.1422806186', 'httponly': False, 'name': '_ga', 'path': '/', 'expiry
': 1485878185, 'expires': 'Tue, 31 Jan 2017 15:56:25 GMT', 'secure':
False, 'domain': '.pythonscraping.com'}, {'value': '1', 'httponly': F
alse, 'name': 'has_js', 'path': '/', 'expiry': 1485878185, 'expires':
 'Tue, 31 Jan 2017 15:56:25 GMT', 'secure': False, 'domain': 'pythons
craping.com'}]
```

Для работы с cookies вы можете вызывать функции `delete_cookie()`, `add_cookie()` и `delete_all_cookies()`. Кроме того, вы можете сохранить cookies, чтобы потом использовать в других веб-скраперах. Ниже приводится пример, который даст вам представление о том, как эти функции можно использовать вместе:

```
from selenium import webdriver

driver = webdriver.PhantomJS(executable_path='<Path to Phantom JS>')
driver.get("http://pythonscraping.com")
driver.implicitly_wait(1)
print(driver.get_cookies())

savedCookies = driver.get_cookies()

driver2 = webdriver.PhantomJS(executable_path='<Path to Phantom JS>')
driver2.get("http://pythonscraping.com")
driver2.delete_all_cookies()
for cookie in savedCookies:
    driver2.add_cookie(cookie)

driver2.get("http://pythonscraping.com")
driver.implicitly_wait(1)
print(driver2.get_cookies())
```

В этом примере первый WebDriver извлекает сайт, выводит cookies, а затем сохраняет их в переменной `savedCookies`. Второй WebDriver загружает тот же самый сайт (техническое примечание: он должен сначала загрузить веб-сайт, чтобы Selenium идентифицировала веб-сайт, которому принадлежат cookies, даже если загрузка веб-сайта не несет ничего полезного для нас), удаляет все его cookies и заменяет их сохраненными cookies из первого Webdriver. При повторной загрузке страницы временные метки, коды и другая информация в обоих наборах cookies должны быть одинаковыми. Согласно Google Analytics, этот второй WebDriver теперь идентичен первому.

Время решает все

Некоторые хорошо защищенные сайты, возможно, не дадут вам отправить веб-формы или взаимодействовать с сайтом, если вы делаете это слишком быстро. Даже если эти функции безопасности не используются, скачивание больших объемов информации с сайта со значительно большей скоростью, чем это может сделать обычный человек, – верный способ обратить на себя внимание и оказаться заблокированным.

Поэтому, несмотря на то что многопоточное программирование – это, возможно, отличный способ загружать страницы гораздо быстрее, позволяющий обрабатывать данные в одном потоке и при этом многократно загружать страницы в другом, оно не годится для написания хороших скраперов. Вы всегда должны пытаться свести к минимуму загрузки отдельных страниц и запросы данных. Если есть

возможность, попробуйте сделать паузу между ними на несколько секунд, даже если для этого потребуется написать дополнительный код:

```
time.sleep(3)
```

Хотя веб-скрапинг часто подразумевает нарушение правил и границ в целях получения данных, есть одно правило, которое вам не нужно нарушать. Чрезмерное потребление ресурсов сервера поставит вас не только в юридически уязвимое положение, но и может нарушить работу небольших сайтов или вызвать их падение. С этической точки зрения вывод сайтов из строя трактуется однозначно: это просто неправильно. Поэтому следите за вашей скоростью!

Общие функции безопасности, используемые веб-формами

Существует масса тестов, которые использовались на протяжении многих лет и продолжают использоваться с разной степенью успеха, чтобы провести отличие между веб-скраперами и людьми. Нет ничего страшного, если бот скачает некоторые статьи и посты в блоге, которые и так были выложены в публичный доступ, однако если бот создает тысячи учетных записей пользователей и начинает отправлять спам участникам Вашего сайта, его действия уже становятся серьезной проблемой. Веб-формы, особенно формы регистрации и авторизации, представляют значительную угрозу с точки зрения безопасности и вычислительных затрат, если они не защищают от ботов, поэтому главная задача многих владельцев сайтов (или, по крайней мере, они так думают) – попытаться ограничить доступ к сайту.

Меры безопасности, используемые веб-формами, могут представлять серьезную проблему для веб-скраперов.

Имейте в виду, что здесь приведен лишь частичный обзор некоторых мер безопасности, с которыми вы можете столкнуться, создавая автоматизированных ботов для обработки этих веб-форм. Посмотрите главу 11, посвященную работе с CAPTCHA и обработке изображений, а также главу 14, посвященную заголовкам и IP-адресам, для получения более подробной информации о работе с хорошо защищенными веб-формами.

Значения полей скрытого ввода

«Скрытые» поля HTML-форм применяются для того, чтобы значение поля использовалось браузером, но при этом было скрыто от пользо-

вателя (за исключением тех случаев, когда пользователь просматривает исходный код сайта). По мере роста популярности cookies с точки зрения хранения и отправки параметров пользователя, скрытые поля на некоторое время исчезли из виду, пока им не была поставлена новая великолепная задача – помешать скраперам обрабатывать веб-формы.

На рис. 12.1 показан пример этих скрытых полей для страницы входа Facebook. Несмотря на то что форма содержит только три видимых поля (имя пользователя, пароль и кнопка отправки), форма передает большое количество информации на сервер в скрытом виде.

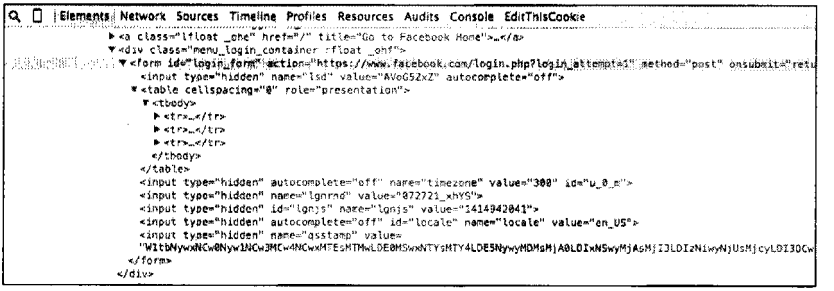


Рис. 12.1 ❖ Форма входа Facebook имеет немало скрытых полей

Есть два основных способа использовать скрытые поля для предотвращения веб-скрапинга: поле может быть заполнено случайно сгенерированным значением на странице формы, которое сервер ожидает для отправки на страницу обработки. Если это значение отсутствует, то сервер может обоснованно предположить, что отправка осуществлена не человеком со страницы формы, а ботом непосредственно на страницу обработки. Лучший способ обойти эту меру – выполнить скрапинг страницы формы, зафиксировать случайно сгенерированное значение, а затем отправить на страницу обработки.

Второй метод – это «горшочек с медом» («honey pot»). Если форма содержит скрытое поле с безобидным названием, например «username» или «email address», простенько написанный бот, возможно, заполнит поле и попытается отправить его, независимо от того, скрыто это поле от пользователя или нет. Любые скрытые поля с заполненными значениями (или значениями, которые отличаются от значений по умолчанию на странице отправки формы) должны игнорироваться, и пользователя, возможно, даже заблокируют.

Короче говоря: иногда необходимо проверить страницу с формой, чтобы убедиться в том, что вы не пропустили каких-либо значений,

ожидаемых сервером. Если вы увидели несколько скрытых полей, часто с длинными случайно сгенерированными строковыми значениями, вполне вероятно, что веб-сервер будет проверять их наличие при отправке формы. Кроме того, веб-сервер может проверить уникальность значений полей, их «срок давности» (это исключает возможность просто записать их в скрипте и использовать снова и снова в течение долгого времени) или одновременно и то, и другое.

Обходим «горшочки с медом»

Хотя CSS по большей части упрощает жизнь, когда дело касается того, чтобы отделить полезную информацию от бесполезной (например, считав теги `id` и `class`), он иногда может представлять проблему для веб-скраперов. Если поле веб-формы скрыто от пользователя с помощью CSS, разумно предположить, что обычный пользователь, посетивший сайт, не сможет заполнить его, потому что оно не отображается в браузере. Если форма *заполнена*, то, скорее всего, здесь поработал бот и `POST`-запрос будет отклонен.

Это относится не только к формам, но и к ссылкам, изображениям, файлам и любому другому элементу сайта, который может быть прочитан ботом, но скрыт от обычного пользователя, заходящего на сайт через браузер. Переход по «скрытой» ссылке на сайте может легко вызвать серверный скрипт, который заблокирует IP-адрес пользователя, выкинет его с сайта или предпримет другое действие, чтобы предотвратить дальнейший доступ к сайту. На самом деле многие модели функционирования сайта построены именно на этом принципе.

Возьмем, к примеру, страницу, расположенную по адресу *http://bit.ly/1GP4sbz*. Эта страница содержит две ссылки, одна скрыта с помощью CSS, а другая видна. Кроме того, на ней размещена форма с двумя скрытыми полями:

```
<html>
<head>
    <title>A bot-proof form</title>
</head>
<style>
    body {
        overflow-x:hidden;
    }
    .customHidden {
        position:absolute;
        right:50000px;
    }
```

```
</style>
<body>
    <h2>A bot-proof form</h2>
    <a href=
     "http://pythonscraping.com/dontgohere" style="display:none;">Go here!</a>
    <a href="http://pythonscraping.com">Click me!</a>
    <form>
        <input type="hidden" name="phone" value="valueShouldNotBeModified"/><p/>
        <input type="text" name="email" class="customHidden"
                  value="intentionallyBlank"/><p/>
        <input type="text" name="firstName"/><p/>
        <input type="text" name="lastName"/><p/>
        <input type="submit" value="Submit"/><p/>
   </form>
</body>
</html>
```

Эти три элемента скрыты от пользователя тремя различными способами:

- первая ссылка скрыта с помощью простого атрибута CSS `display:none`;
- поле «phone» является полем скрытого ввода;
- поле «email» скрыто путем перемещения его на 50 000 пикселей вправо.

К счастью, поскольку Selenium фактически выполняет рендеринг посещаемых страниц, она может отличить элементы, которые визуально присутствуют на странице, от тех, которые на странице не отображаются. Присутствие элемента на странице может быть определено с помощью функции `is_displayed()`.

Например, код, приведенный ниже, извлекает ранее описанную страницу, ищет скрытые ссылки и поля ввода:

```
from selenium import webdriver
from selenium.webdriver.remote.webelement import WebElement

driver = webdriver.PhantomJS(executable_path='')
driver.get("http://pythonscraping.com/pages/itsatrap.html")
links = driver.find_elements_by_tag_name("a")
for link in links:
    if not link.is_displayed():
        print("The link "+link.get_attribute("href")+" is a trap")

fields = driver.find_elements_by_tag_name("input")
for field in fields:
    if not field.is_displayed():
        print("Do not change value of "+field.get_attribute("name"))
```

Selenium находит все скрытые поля, генерируя следующий вывод:

```
The link http://pythonscraping.com/dontgohere is a trap
Do not change value of phone
Do not change value of email
```

Несмотря на то что вы, вероятно, не собираетесь ходить по любым скрытым ссылкам, найденным на сайте, убедитесь, что отправляете заранее заполненные скрытые значения формы (или заставляете Selenium отправить их за вас) вместе с остальными значениями. Подведем итог: опасно просто игнорировать скрытые поля, и в то же время вы должны быть очень осторожны при работе с ними.

Проверяем скрапер на «человечность»

В этой главе и вообще в этой книге очень много рассказывается о том, как построить скрапер, который меньше всего похож на скрапер и максимально похож на человека. Если веб-сайты продолжают блокировать вас и вы не знаете причины, ниже приведен список, который вы можете использовать для устранения проблем:

- во-первых, если загруженная страница является пустой, содержит пропущенную информацию или, наоборот, не то, что вы ожидали увидеть (или уже видели в вашем собственном браузере), скорее всего, это вызвано тем, что на сайте выполняется JavaScript, создающий страницу. Посмотрите главу 10;
- если вы отправляете форму или осуществляете POST-запрос к веб-сайту, проверьте страницу, чтобы убедиться, что все значения, которые веб-сайт ожидает от вас, заданы и имеют правильный формат. Используйте такой инструмент, как Chrome Network Inspector, чтобы посмотреть, как выглядит фактический POST-запрос, отправляемый на сайт, чтобы убедиться в том, что вы предусмотрели все;
- если вы пытаетесь зайти на сайт и не можете залогиниться или сайт как-то странно ведет себя, проверьте cookies. Убедитесь, что cookies правильно сохраняются во время загрузки страниц и отправляются на сайт при каждом запросе;
- если вы получаете ошибки HTTP-протокола со стороны клиента, особенно ошибки 403 Forbidden, это может означать, что веб-сайт определил вас (ваш IP) как бота и не желает больше принимать запросы. Вам нужно либо дождаться, когда ваш IP-адрес будет удален из списка, либо получить новый IP-адрес

(либо пойти в Starbucks, либо обратиться к главе 14). Чтобы убедиться в том, что вы не будете снова заблокированы, попробуйте следующее:

- убедитесь, что ваши скраперы не перемещаются по сайту слишком быстро. Высокая скорость скрапинга является плохой практикой, которая ложится тяжелым бременем на серверы веб-администратора, может привести вас к неприятностям с законом и является причиной номер один, по которой скраперы попадают в черный список. Добавьте паузы в работу веб-скрапера и оставьте работать на ночь. Помните: лихорадочно писать программы или собирать данные – это признак плохого управления проектами. Прежде всего планируйте, чтобы избежать бардака;
- очевидно: измените заголовки! Некоторые сайты будут блокировать все, что рекламирует себя как скрапер. Скопируйте заголовки вашего собственного браузера, если у вас нет информации о подходящих значениях заголовков;
- убедитесь в том, что вы не нажали или не заполнили ничего такого, что обычный человек, как правило, не сможет нажать или заполнить (вернитесь к разделу «Обходим «горшочки с медом» для получения дополнительной информации);
- если вы столкнулись со множеством препятствий, пытаясь получить доступ, рассмотрите возможность связаться с администратором сайта, чтобы сообщить ему о своих намерениях. Попробуйте написать ему на электронную почту *webmaster@<domain name>* или *admin@<domain name>*, чтобы получить разрешение на скрапинг веб-сайта. Админы – тоже люди!

Глава 13

Тестирование вашего сайта с помощью скраперов

При работе с веб-проектами, которые используют широкий стек технологий, часто тестируется только внутренняя часть этого стека. На сегодняшний день большинство языков программирования (в том числе Python) предлагают определенную платформу тестирования, однако «лицевая часть» (front-end) веб-сайта часто выпадает из этих автоматизированных тестов, хотя она, возможно, является единственной клиентской частью проекта.

Частично проблема заключается в том, что веб-сайты часто представляют собой смесь различных языков разметки и языков программирования. Вы можете написать модульные тесты для JavaScript, но это бесполезно, если HTML, с которым он взаимодействует, изменился таким образом, что JavaScript не выполняет планируемого действия на странице, даже если он работает правильно.

Проблему тестирования «лицевой части» веб-сайта часто решают постфактум или поручают программистам нижнего звена, вооруженным в лучшем случае контрольным списком и багтрекером. Однако, приложив чуть больше усилий, вы можете вместо контрольного списка использовать набор модульных тестов, а вместо глаз – веб-скрапер.

Представьте себе: для нашего веб-проекта мы используем технологию «разработки через тестирование». Проводим ежедневные тесты, чтобы убедиться, что все элементы веб-интерфейса функциониру-

ют должным образом. Каждый раз, когда кто-либо добавляет новую функцию веб-сайта или изменяет расположение элемента, запускается набор тестов. В этой главе мы рассмотрим основы тестирования и способы тестирования всех видов веб-сайтов (начиная от простых и заканчивая сложными) с помощью скраперов, написанных на языке Python.

Введение в тестирование

Если вы раньше никогда не писали тесты для вашего программного кода, сейчас самое подходящее время сделать это. Воспользовавшись набором тестов, который можно запустить для проверки правильности вашего кода (по крайней мере, с точки зрения написанных тестов), вы сэкономите время и нервы и легко обновите сайт в очередной раз.

Что такое модульные тесты?

Слова *тест* (*test*) и *модульный тест* (*unit test*) часто используются как синонимы. Нередко, когда программисты говорят о «написании тестов», они на самом деле имеют в виду «написание модульных тестов». С другой стороны, некоторые программисты под написанием модульных тестов подразумевают какие-то другие виды тестов.

Хотя у каждой компании, как правило, свои определения и практики тестирования, обычно модульный тест имеет следующие характеристики:

- каждый модульный тест проверяет один функциональный аспект компонента. Например, он гарантирует, что при снятии с банковского счета отрицательной суммы в долларах будет сгенерировано соответствующее сообщение об ошибке.
  Часто модульные тесты сгруппированы вместе в одном классе на основе тестируемого компонента. Вы можете запустить тест, чтобы проверить снятие с банковского счета отрицательных сумм, после которого запускается модульный тест, проверяющий правильность осуществления транзакций по этому банковскому счету в целом;
- каждый модульный тест можно запустить полностью автономно, и функции setup и teardown, необходимые для модульного тестирования, должны выполняться самим модульным тестом. Кроме того, модульные тесты не должны зависеть от результатов выполнения других тестов, и они должны успешно выполняться в любом порядке;

- каждый модульный тест обычно содержит, по крайней мере, одно *утверждение* (*assertion*). Например, модульный тест может утверждать, что ответом на 2 + 2 будет 4. Иногда модульный тест завершается отказом.
  Например, при возникновении исключения модульный тест может прерваться, но если все идет гладко, то по умолчанию он будет выполнен полностью;
- Модульные тесты отделены от основного кода. Несмотря на то что они обязательно должны импортировать и использовать тестируемый код, их, как правило, хранят в отдельных классах и каталогах.

Хотя есть много других типов тестов, которые можно написать, например интеграционные тесты и проверочные тесты, мы в этой главе в первую очередь сосредоточимся на модульном тестировании. Модульные тесты стали чрезвычайно популярными наряду с последними тенденциями в сторону «разработки через тестирование». Кроме того, компактность и гибкость модульных тестов позволяют легко использовать их в качестве примеров, а Python обладает некоторыми встроенными возможностями модульного тестирования, как мы увидим в следующем разделе.

Питоновский модуль unittest

Модуль `unittest` входит в стандартную библиотеку Python. Просто импортируем `unittest` и объявляем `unittest.TestCase`, который выполняет следующее:

- задает функции `setUp` и `tearDown`, которые запускаются до и после каждого модульного теста;
- задает несколько типов проверки утверждений (если проверка утверждения завершается успешно, тест проходит, в противном случае тест завершается с ошибкой);
- запускает все функции, которые начинаются с префикса `test_` в качестве модульных тестов, и игнорирует функции, которые не рассматриваются в качестве тестов.

Ниже приводится очень простой модульный тест, написанный на Python, который проверяет, что 2 + 2 = 4:

```
import unittest

class TestAddition(unittest.TestCase):
    def setUp(self):
```

```
        print("Setting up the test")

    def tearDown(self):
        print("Tearing down the test")

    def test_twoPlusTwo(self):
        total = 2+2
        self.assertEqual(4, total);

if __name__ == '__main__':
    unittest.main()
```

Несмотря на то что setUp и tearDown не выполняют здесь никаких полезных действий, они даны в целях иллюстрации. Обратите внимание, что эти функции выполняются до и после каждого отдельного теста, а не до и после всех тестов в классе.

Тестирование Википедии

Протестировать «лицевую часть» вашего сайта (за исключением JavaScript, который мы рассмотрим далее) очень просто, объединив питоновскую библиотеку unittest с веб-скрапером:

```
from urllib.request import urlopen
from bs4 import BeautifulSoup
import unittest

class TestWikipedia(unittest.TestCase):
    bsObj = None
    def setUpClass():
        global bsObj
        url = "http://en.wikipedia.org/wiki/Monty_Python"
        bsObj = BeautifulSoup(urlopen(url))

    def test_titleText(self):
        global bsObj
        pageTitle = bsObj.find("h1").get_text()
        self.assertEqual("Monty Python", pageTitle);

    def test_contentExists(self):
        global bsObj
        content = bsObj.find("div",{"id":"mw-content-text"})
        self.assertIsNotNone(content)

if __name__ == '__main__':
    unittest.main()
```

На этот раз у нас два теста: первый тест проверяет, соответствует ли название страницы ожидаемому названию «Monty Python», а второй убеждается в том, что страница имеет контент, заключенный в DIV.

Обратите внимание, содержимое страницы загружается только один раз, и глобальный объект bsObj является общим для тестов. Это достигается за счет использования функции setUpClass модуля unittest, которая выполняется лишь один раз в начале определения класса (в отличие от setUp, которая запускается перед каждым отдельным тестом). Использование setUpClass вместо setUp избавляет от ненужных загрузок страницы, мы можем собрать контент сразу и протестировать его.

Хотя тестирование отдельной страницы, возможно, не покажется вам таким захватывающим или интересным, как вы помните из главы 3, создать веб-скрапер, который может итеративно перемещаться по всем страницам сайта, довольно легко. Что произойдет, если мы объединим веб-скрапер с модульным тестом, который проверит утверждения на каждой странице?

Есть масса способов запускать тесты итеративно, но мы должны соблюдать осторожность, загружая по одной странице для каждого набора тестов и избегая одновременной загрузки больших объемов информации в память. Код, приведенный ниже, как раз это и делает:

```
class TestWikipedia(unittest.TestCase):
    bsObj = None
    url = None

    def test_PageProperties(self):
        global bsObj
        global url

        url = "http://en.wikipedia.org/wiki/Monty_Python"
        #Тестируем первые 100 страниц
        for i in range(1, 100):
            bsObj = BeautifulSoup(urlopen(url))
            titles = self.titleMatchesURL()
            self.assertEquals(titles[0], titles[1])
            self.assertTrue(self.contentExists())
            url = self.getNextLink()
        print("Done!")

    def titleMatchesURL(self):
        global bsObj
        global url
        pageTitle = bsObj.find("h1").get_text()
        urlTitle = url[(url.index("/wiki/")+6):]
        urlTitle = urlTitle.replace("_", " ")
        urlTitle = unquote(urlTitle)
        return [pageTitle.lower(), urlTitle.lower()]
```

```
    def contentExists(self):
        global bsObj
        content = bsObj.find("div",{"id":"mw-content-text"})
        if content is not None:
            return True
        return False

    def getNextLink(self):
        #Возвращает случайную ссылку на страницу, используя метод из главы 5

if __name__ == '__main__':
    unittest.main()
```

Нужно отметить несколько вещей. Во-первых, в этом классе выполняется лишь один фактический тест. Остальные функции с технической точки зрения являются лишь вспомогательными функциями, несмотря на то что они делают бóльшую часть вычислений, чтобы определить, прóходит ли тест. Поскольку тестовая функция выполняет проверки утверждений, результаты теста передаются обратно в нее, где эти проверки и происходят.

Кроме того, `contentExists` возвращает булево значение, а `titleMatchesURL` возвращает сами значения для проверки. Чтобы понять, почему нужно возвращать значения, а не просто булево значение, сравните результат булевой проверки утверждений:

```
======================================================================
FAIL: test_PageProperties (__main__.TestWikipedia)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "15-3.py", line 22, in test_PageProperties
    self.assertTrue(self.titleMatchesURL())
AssertionError: False is not true
```

с результатом проверки `assertEquals`:

```
======================================================================
FAIL: test_PageProperties (__main__.TestWikipedia)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "15-3.py", line 23, in test_PageProperties
    self.assertEquals(titles[0], titles[1])
AssertionError: 'lockheed u-2' != 'u-2 spy plane'
```

Какой из них легче отладить?

В данном случае ошибка происходит из-за редиректа, когда статья *http://bit.ly/1Jcc0qF* перенаправляет нас на статью под названием «Lockheed U-2».

Тестирование с помощью Selenium

Как и Ajax, описанный в главе 10, JavaScript представляет определенные трудности при выполнении тестирования веб-сайта. К счастью, Selinium имеет отличную платформу для работы с особо сложными веб-сайтами. На самом деле эта библиотека была изначально разработана для тестирования веб-сайтов!

Несмотря на то что питоновский модуль unittests и модульные тесты Selenium написаны на одном и том же языке, они не удивлении имеют мало общего. Selenium не требует, чтобы ее модульные тесты задавались в качестве функций внутри классов, проверки утверждений не требуют скобок, и тесты проходят молча, лишь генерируя определенные сообщения об ошибке:

```
driver = webdriver.PhantomJS()
driver.get("http://en.wikipedia.org/wiki/Monty Python")
assert "Monty Python" in driver.title
driver.close()
```

При запуске этот тест не сгенерирует никакого вывода.

Таким образом, писать тесты Selenium проще, чем unittests, а проверки утверждений можно даже интегрировать в обычный код, когда в случае невыполнения какого-либо условия желательно прекратить выполнение кода.

Взаимодействие с сайтом

Недавно я хотела связаться с местной коммерческой фирмой через контактную форму ее веб-сайта, но обнаружила, что HTML-форма не работает: ничего не произошло, когда я нажала на кнопку отправки. Проведя небольшое расследование, я увидела, что они использовали простую форму mailto, которая предназначалась для отправки им электронного письма с заполненными полями формы. К счастью, я смогла воспользоваться информацией, чтобы отправить им письмо по электронной почте, объяснила проблему с их формой и наняла их, несмотря на возникшие технические проблемы.

Если бы мне понадобилось написать традиционный скрапер, который использовал или тестировал эту форму, мой скрапер, скорее всего, просто скопировал бы макет формы и отправил его по электронной почте напрямую – вообще минуя форму. Как я могу протестировать форму и убедиться, что она отлично работает, *с помощью* браузера?

Хотя в предыдущих главах мы уже рассматривали перемещение по ссылкам, отправку форм, а также другие виды взаимодействия,

по сути, все, что мы делали, предназначалось для *обхода* интерфейса браузера, а не для его использования. С другой стороны, Selenium может в буквальном смысле ввести текст, нажать кнопки и сделать все с помощью браузера (в данном с помощью браузера PhantomJS, у которого нет графического интерфейса), и обнаружить такие вещи, как неработающую форму, неправильно составленный код JavaScript, ошибки HTML-кода, а также другие проблемы, которые могут поставить в тупик реальных клиентов.

В данном виде тестирования ключевым является понятие `elements`. Этот объект кратко освещался в главе 10 и возвращается с помощью вызовов типа:

```
usernameField = driver.find_element_by_name('username')
```

Подобно тому, как вы осуществляете определенные действия над различными элементами веб-сайта в вашем браузере, точно так же и Selenium может выполнить большое количество действий над любым элементом. Среди них:

```
myElement.click()
myElement.click_and_hold()
myElement.release()
myElement.double_click()
myElement.send_keys_to_element("content to enter")
```

Кроме выполнения однократной операции над элементом, строки кода, осуществляющие действия, можно объединить в *цепочки действий* (*action chains*), которые можно сохранить и выполнить один или несколько раз в программе. Цепочки действий полезны тем, что они являются удобным способом записи длинных последовательностей, состоящих из нескольких операций, при этом с функциональной точки зрения они идентичны явному вызову действия над элементом, как и в предыдущих примерах.

Чтобы увидеть разницу между действием и цепочкой действий, взгляните на страницу формы на http://bit.ly/1AGKPRU (которая ранее была использована в качестве примера в главе 9). Мы можем заполнить форму и отправить ее следующим образом:

```
from selenium import webdriver
from selenium.webdriver.remote.webelement import WebElement
from selenium.webdriver.common.keys import Keys
from selenium.webdriver import ActionChains

driver = webdriver.PhantomJS(executable_path='<Path to Phantom JS>')
```

```
driver.get("http://pythonscraping.com/pages/files/form.html")

firstnameField = driver.find_element_by_name("firstname")
lastnameField = driver.find_element_by_name("lastname")
submitButton = driver.find_element_by_id("submit")

### МЕТОД 1 ###
firstnameField.send_keys("Ryan")
lastnameField.send_keys("Mitchell")
submitButton.click()
###################

### МЕТОД 2 ###
actions = ActionChains(driver).click(firstnameField).send_keys("Ryan")
                               .click(lastnameField).send_keys("Mitchell")
                               .send_keys(Keys.RETURN)
actions.perform()
###################

print(driver.find_element_by_tag_name("body").text)

driver.close()
```

Метод 1 вызывает send_keys для двух полей, затем нажимает кнопку отправки, тогда как метод 2 использует цепочку действий (нажимает кнопку и вводит текст в каждом поле), которая выполняется последовательно после вызова метода perform. Если используется первый или второй метод, этот скрипт приведет к одинаковым результатам и выведет строку:

```
Hello there, Ryan Mitchell!
```

Помимо объектов, которые используются для обработки команд, существует еще одно отличие между этими двумя методами: обратите внимание, что первый метод нажимает кнопку «submit», в то время как второй использует клавишу «return» для отправки формы, когда текстовое поле заполнено. Поскольку одно и то же действие можно представить в виде различных последовательностей событий, точно так же и в Selenium одно и то же действие можно выполнить по-разному.

Перетаскивание элементов

Нажатие кнопок и ввод текста – это здорово, но подлинная мощь Selenium заключается в ее способности работать с относительно новыми формами веб-взаимодействия. Selenium позволяет легко работать с интерфейсами, реализующими перетаскивание элементов (drag-and-drop). Чтобы использовать функции перетаскивания, вам

требуется указать исходный элемент (элемент, который нужно переместить), а также либо смещение элемента, либо целевой элемент, в который нужно перетащить исходный элемент.

Демонстрационная страница, расположенная по адресу *http://bit.ly/1GP52py*, представляет собой пример такого интерфейса:

```
from selenium import webdriver
from selenium.webdriver.remote.webelement import WebElement
from selenium.webdriver import ActionChains

driver = webdriver.PhantomJS(executable_path='<Path to Phantom JS>')
driver.get('http://pythonscraping.com/pages/javascript/draggableDemo.html')

print(driver.find_element_by_id("message").text)

element = driver.find_element_by_id("draggable")
target = driver.find_element_by_id("div2")
actions = ActionChains(driver)
actions.drag_and_drop(element, target).perform()

print(driver.find_element_by_id("message").text)
```

Выводятся два сообщения из div-блока `message` на демонстрационной странице. Первое сообщение гласит:

```
Prove you are not a bot, by dragging the square from the blue area to the red area!
```

Затем сразу после выполнения задания снова выводится сообщение, которое теперь выглядит так:

```
You are definitely not a bot!
```

Конечно же, как и предполагает демонстрационная страница, перетаскивание элементов, служащее доказательством того, что вы не являетесь ботом, является часто используемой темой во многих тестах CAPTCHA. Хотя ботам все-таки удается перетащить объекты, если дать им время (это всего лишь вопрос нажатия, удерживания и перемещения), так или иначе идея «перетащите это» как способ отличить человека от бота просто так не умрет.

Кроме того, эти CAPTCHA-библиотеки с перетаскиванием элементов редко используют задачи, сложные для ботов, вроде «перетащите картинку с котенком на картинку с коровой» (для выполнения этой инструкции вам потребуется найти картинку с «котенком» и картинку с «коровой»). Вместо этого они часто используют расположение цифр по возрастанию или убыванию или какую-нибудь другую довольно тривиальную задачу, как та, что была приведена в предыдущем примере.

Разумеется, преимущество тестов CAPTCHA заключается в том, что существует такое большое количество различных комбинаций и вероятность угадать их настолько мала, что, скорее всего, никто не будет тратить время на создание бота, который сможет решить все эти задачи. В любом случае этого примера должно быть достаточно, чтобы показать, почему вы никогда не должны использовать эту технику для крупных веб-сайтов.

Снятие скриншотов

Помимо обычных возможностей тестирования, у Selenium есть одна интересная хитрость, которая может немного упростить ваше тестирование (или произвести впечатление на босса), – скриншоты. Да, фотосвидетельство можно получить прямо в модульных тестах, фактически не нажимая клавишу PrtScn:

```
driver = webdriver.PhantomJS()
driver.implicitly_wait(5)
driver.get('http://www.pythonscraping.com/')
driver.get_screenshot_as_file('tmp/pythonscraping.png')
```

Этот скрипт переходит на страницу *http://pythonscraping.com*, делает пятисекундную паузу для выполнения любого скрипта JavaScript, затем сохраняет скриншот домашней страницы в локальной папке *tmp* (папка должна уже существовать).

Unittest или Selenium?

Синтаксическая строгость и детализированность питоновского модуля unittest, возможно, будут привлекательны для большинства платформ тестирования, в то время как гибкость и мощность тестов Selenium, вероятно, как нельзя лучше подходят для проверки некоторых функций веб-сайта. Так какой из этих инструментов использовать?

Секрет в том, что вы не должны выбирать. Selenium удобно использовать для получения определенной информации о веб-сайте, тогда как unittest может определить, соответствует ли эта информация критериям прохождения теста. Нет такой причины, в силу которой вы не можете импортировать инструменты Selenium в питоновский unittest, сочетая лучшее из обоих миров.

Например, скрипт, приведенный ниже, создает модульный тест для интерфейса сайта с поддержкой перетаскивания элементов. Тест проверяет выдачу сообщения «You are not a bot!» после того, как один элемент перемещен к другому:

```
from selenium import webdriver
from selenium.webdriver.remote.webelement import WebElement
from selenium.webdriver import ActionChains
import unittest

class TestAddition(unittest.TestCase):
    driver = None
    def setUp(self):
        global driver
        driver = webdriver.PhantomJS(executable_path='<Path to Phantom JS>')
        url = 'http://pythonscraping.com/pages/javascript/draggableDemo.html'
        driver.get(url)

    def tearDown(self):
        print("Tearing down the test")

    def test_drag(self):
        global driver
        element = driver.find_element_by_id("draggable")
        target = driver.find_element_by_id("div2")
        actions = ActionChains(driver)
        actions.drag_and_drop(element, target).perform()

        self.assertEqual("You are definitely
                         not a bot!", driver.find_element_by_id(
                         "message").text)

if __name__ == '__main__':
    unittest.main()
```

Практически все элементы сайта можно проверить с помощью комбинации питоновского unittest и Selenium. На самом деле, использовав дополнительно некоторые библиотеки обработки изображений, о которых рассказывалось в главы 11, вы можете даже сделать скриншот веб-сайта и протестировать пиксель за пикселем его контент!

Глава 14

Скрапинг с помощью удаленных серверов

То, что эта глава является последней, в какой-то мере логично. До сих пор мы запускали все приложения для Python из командной строки, не выходя за рамки использования наших домашних компьютеров. Конечно, вы можете установить MySQL в попытке воссоздать среду реального сервера. Но это не равнозначные вещи.

В этой главе я расскажу о том, как можно запускать скрипты с различных машин или даже с разных IP-адресов на вашей собственной машине. Несмотря на то что вы, возможно, сейчас захотите пропустить этот шаг как *ненужный*, вы удивитесь, насколько удобно станет работать с уже имеющимися инструментами (например, с личным веб-сайтом на платном хостинге) и насколько станет легкой жизнь, когда вы перестанете запускать скраперы Python с ноутбука.

Зачем использовать удаленные серверы?

Несмотря на то что использование удаленного сервера выглядит очевидным шагом при запуске веб-приложения, предназначенного для широкой аудитории, инструменты, которые мы создаем для решения наших собственных задач, чаще всего работают локально. Люди, которые предпочитают пользоваться удаленной платформой, как правило, обосновывают свое решение двумя главными причинами: потребностью в более высокой функциональности и гибкости, а также необходимостью использования альтернативного IP-адреса.

Как избежать блокировки IP-адреса

При создании веб-скрапера работает правило: все можно подделать. вы можете отправить электронное письмо с адреса, которым не владеете, двигать мышью из командной строки или даже напугать веб-администратора, посетив его веб-сайт с помощью Internet Explorer 5.0.

Единственное, что нельзя подделать, – это ваш IP-адрес. Любой желающий может отправить вам письмо с обратным адресом: «The President, 1600 Pennsylvania Avenue Northwest, Washington, DC 20500». Однако, если письмо отправлено из Альбукерке, Нью-Мексико, вы можете быть уверены в том, что отправитель не имеет ничего общего с Президентом США[1].

Большинство усилий, направленных на то, чтобы помешать скраперам получить доступ к веб-сайтам, сосредоточено на определении разницы между людьми и ботами. Блокировка IP-адреса немного напоминает фермера, который, отказавшись от опрыскивания пестицидами, решил просто сжечь поле. Это отчаянный, но очень эффективный метод просто отклонить пакеты, отправленные с назойливых IP-адресов. Однако у этого решения есть ряд проблем:

- со списками доступа IP сложно работать. Несмотря на то что крупные сайты чаще всего используют свои собственные программы, автоматизирующие некоторые рутинные операции с этими списками (боты, блокирующие ботов!), кто-то должен хотя бы иногда проверять их или, по крайней мере, отслеживать их рост во избежание проблем;
- каждый адрес немного увеличивает время обработки при получении пакетов, поскольку сервер должен проверить принятые пакеты по списку и решить, следует одобрить их или отклонить. Большое количество адресов, помноженное на большое количество пакетов, может значительно увеличить время обработки. Чтобы не потерять скорость обработки и гибкость, администраторы часто группируют эти IP-адреса в блоки и задают правила типа «все 256 адресов в этом диапазоне являются

[1] С технической точки зрения IP-адреса можно подменить в исходящих пакетах. Этот метод используется в распределенных DDoS-атаках, когда злоумышленники не заботятся о получении возвращаемых пакетов (которые, если их передать, будут отправлены на неправильный адрес). Но веб-скрапинг по определению является процессом, в котором требуется ответ от веб-сервера, поэтому мы считаем, что IP-адреса – это единственная вещь, которую нельзя подделать.

заблокированными», если уже есть несколько злоумышленников с очень схожими IP-адресами. Это приводит нас к третьему пункту;

- блокировка IP-адреса может привести к непредвиденным последствиям. Например, когда я училась в Инженерном колледже имени Франклина Олина, один студент написал программу, которая пыталась подтасовать голоса, поданные за популярный контент на сайте *http://digg.com* (это было еще до того, как Reddit стал популярным). Блокировка одного IP-адреса привела к тому, что целое общежитие не могло получить доступ к сайту. Студент просто перенес свою программу на другой сервер, тогда как Digg лишился значительной части обычных посетителей, составляющих его основную аудиторию.

Несмотря на все недостатки, у администраторов серверов блокировка IP-адреса остается наиболее распространенным методом защиты серверов от подозрительных веб-скреперов.

Переносимость и расширяемость

Некоторые задачи просто слишком обременительны для домашнего компьютера и интернет-соединения. Вам не нужно создавать большую нагрузку на какой-то отдельный веб-сайт, вы можете собрать информацию с различных сайтов, воспользовавшись гораздо большей пропускной способностью и объемом хранения, чем это могут позволить ваши текущие настройки.

Кроме того, избавив процессор от интенсивных вычислений, вы можете высвободить ресурсы Вашего домашнего компьютера для решения более важных задач (кто хочет сыграть в World Of Warcraft?). Вам не придется беспокоиться о поддержании ресурсов на должном уровне и подключении к Интернету (запустите приложение, сидя в Starbucks, положите ноутбук в сумку и идите гулять, зная, что все по-прежнему прекрасно работает), и вы можете получить доступ к собранным данным в любом месте, где есть подключение к Интернету.

Если вы используете приложение, которое требует столько ресурсов, что одного внушительного вычислительного сервиса Amazon будет недостаточно, посмотрите в сторону *распределенных вычислений* (*distributed computing*). Это позволяет нескольким компьютерам работать над вашими задачами параллельно. Например, один компьютер выполняет краулинг одного набора сайтов, а другой компьютер выполняет крулинг второго набора сайтов, и оба сохраняют собранные данные в одной и той же базе данных.

Конечно, как уже отмечалось в предыдущих главах, многие могут воспроизвести то, что делает поиск Google, но очень немногие могут повторить масштаб этого поиска. Распределенные вычисления – очень большая область компьютерных наук, которая выходит за рамки этой книги. Однако научиться запускать приложение на удаленном сервере – это необходимый первый шаг, и вы, возможно, удивитесь тому, на что компьютеры способны в наши дни.

Tor

Сеть Onion Router, более известная под аббревиатурой *Tor*, представляет собой сеть серверов добровольцев, созданную для адресации и переадресации трафика через большое количество сетевых узлов (луковых маршрутизаторов), чтобы скрыть свое происхождение. Данные шифруются перед входом в сеть таким образом, чтобы гарантировать анонимность соединения. Кроме того, хотя входящие и исходящие сообщения любого сетевого узла могут быть скомпрометированы, чтобы расшифровать истинные начальные и конечные пункты соединения, необходимо получить подробную информацию о входящих и исходящих сообщениях для *всех* сетевых узлов, составляющих цепочку соединений, – а это практически невыполнимый подвиг.

Tor обычно используется сотрудниками правозащитных организаций и политическими информаторами для общения с журналистами и получает значительную часть своего финансирования со стороны правительства США. Конечно, он также широко используется для осуществления незаконной деятельности и таким образом остается объектом пристального надзора со стороны государства (несмотря на то что до настоящего времени этот надзор имел лишь смешанный успех).

Ограничения анонимности Tor

Хотя причина, по которой мы используем Tor в этой книге, – это изменение нашего IP-адреса, а не достижение полной анонимности как таковой, стоит уделить внимание некоторым преимуществам и ограничениям анонимизации трафика, обеспечиваемой с помощью Tor.

Хотя вы можете подумать, что, используя Tor, ваш исходящий IP-адрес, передаваемый на веб-сервер, – это уже не тот IP-адрес, по которому вас могут отследить, любая информация, которой вы поделитесь с этим веб-сервером, может раскрыть вас. Например, если вы войдете в ваш собственный аккаунт Gmail, а затем выполните компрометирующий поиск в Google, результаты этого поиска теперь могут быть привязаны к вашей личности.

Однако, помимо очевидных вещей, даже сам вход в Tor может угрожать вашей анонимности. В декабре 2013 года студент Гарварда, пытаясь сорвать выпускные экзамены, отправил по электронной почте письмо об угрозе взрыва в университете, воспользовавшись сетью Tor и анонимным аккаунтом электронной почты. Когда IT-специалисты Гарварда изучили логи, они обнаружили Tor-трафик, который исходил только от одного компьютера, зарегистрированного на определенного студента, и был сгенерирован в то же самое время, когда было отправлено письмо об угрозе взрыва. Несмотря на то что они не смогли определить возможное место назначения этого трафика (только то, что он был отправлен через Tor), того факта, что время генерации трафика совпадало со временем отправки письма и лишь один компьютер в тот момент использовал Tor, уже было достаточно для изобличения и судебного преследования студента.

Войти в Tor – это не значит автоматически надеть плащ-невидимку, и Tor не дает вам право делать в Интернете все, что вам угодно. Несмотря на то, он является полезным инструментом, обязательно используйте его осмотрительно, с умом и, конечно же, руководствуясь моральными принципами.

Как мы увидим в следующем разделе, установка и запуск Tor – обязательное требование для использования Python в сети Tor. К счастью, сервис Tor чрезвычайно прост в установке и запуске. Просто зайдите на страницу Tor (*http://bit.ly/1eLkRmv*), скачайте, установите, откройте и подключитесь! Имейте в виду, что при использовании Tor скорость интернет-соединения может показаться более низкой. Наберитесь терпения, возможно, он несколько раз обходит весь мир!

PySocks

PySocks является довольно простым модулем Python, который способен маршрутизировать трафик с помощью прокси-серверов и прекрасно работает в связке с Tor. Вы можете скачать его с сайта или воспользоваться инсталляторами сторонних разработчиков, чтобы установить его.

Хотя в плане документации написано по этому модулю не так много, использовать его очень просто. Во время выполнения кода, приведенного ниже, сервис Tor должен использовать порт 9150 (порт по умолчанию):

```
import socks
import socket
from urllib.request import urlopen

socks.set_default_proxy(socks.SOCKS5, "localhost", 9150)
socket.socket = socks.socksocket
print(urlopen('http://icanhazip.com').read())
```

Веб-сайт http://icanhazip.com показывает только IP-адрес клиента, подключенного к серверу, и может быть использован в целях тестирования. Когда вышеприведенный скрипт запущен, сайт должен показывать другой IP-адрес.

Если вы хотите использовать Selenium и PhantomJS в сети Tor, модуль PySocks вам не нужен вообще – просто убедитесь, что Tor в данный момент работает, и добавьте дополнительные параметры service_args, указав, что Selenium должен подключаться через порт 9150:

```
from selenium import webdriver
service_args = [ '--proxy=localhost:9150', '--proxy-type=socks5', ]
driver = webdriver.PhantomJS(executable_path='<path to PhantomJS>', \
                             service_args=service_args)

driver.get("http://icanhazip.com")
print(driver.page_source)
driver.close()
```

И вновь этот код должен вывести IP-адрес, но не ваш собственный, а тот, что использует в данный момент клиент Tor, запущенный вами.

Удаленный хостинг

Хотя полная анонимность теряется, как только вы решите расплатиться кредитной картой, установка ваших веб-скраперов на удаленный хостинг значительно повысит их скорость. Это обусловлено тем, что вы покупаете время работы на более мощных машинах, чем Ваша собственная, а также тем, что интернет-соединению не нужно проходить через маршрутизаторы сети Tor, чтобы достичь места назначения.

Запуск с аккаунта веб-хостинга

Если вы являетесь владельцем личного или делового сайта, то у вас, скорее всего, уже есть инструменты, позволяющие запустить ваши веб-скраперы с внешнего сервера. Даже при работе с защищенными веб-серверами, где у вас нет доступа к командной строке, можно запускать и прекращать выполнение скриптов с помощью веб-интерфейса.

Если ваш веб-сайт размещен на сервере Linux, там, скорее всего, уже установлен Python. Если ваш хостинг-аккаунт находится на сервере под управлением Windows, вам, возможно, немного не повезло. Вам нужно проверить, установлен ли Python, а если нет, то готов ли администратор сервера установить его.

Значительная часть небольших хостинг-провайдеров предлагает программное обеспечение под названием *cPanel*, которое используется для предоставления базовых услуг по администрированию, а также информации о вашем сайте и подключенных услугах. Если у вас есть доступ к cPanel, убедитесь, что Python готов к запуску на сервере, перейдя в «Apache Handlers» и добавив новый обработчик (если он еще не задан):

```
Handler: cgi-script
Extension(s): .py
```

Эта настройка сообщает Вашему серверу о том, что скрипты Python должны выполняться в виде CGI-скрипта. CGI (расшифровывается как Common Gateway Interface, или Общий интерфейс шлюза) – это *просто любая программа, которая запускается на сервере и динамически генерирует контент*, отображаемый на веб-сайте. Явно определив скрипты Python как CGI-скрипты, вы разрешаете серверу выполнить их, а не просто вывести их в браузере или отправить пользователю ссылку для скачивания.

Просто напишите скрипт Python, загрузите его на сервер и установите права доступа к файлам 755, чтобы разрешить выполнение скрипта. Для выполнения скрипта просто перейдите к месту загрузки скрипта, используя браузер (или лучше напишите скрапер, который сделает это за вас). Если вы беспокоитесь по поводу того, что кто-то получит доступ к Вашему скрипту и запустит его, у вас есть два варианта:

- для хранения скрипта используйте неявный или скрытый URL-адрес и убедитесь, что никогда не ссылались на этот скрипт с помощью каких-либо других доступных URL-адресов, чтобы избежать его индексации поисковыми системами;
- защитите скрипт с помощью пароля, либо запрашивайте пароль или секретный токен перед выполнением скрипта.

Конечно, запуск скрипта Python из сервиса, который сам предназначен для показа веб-сайтов, является своего рода хаком. Вы, вероятно, заметили, что ваш сайт web-scraper.com немного медленнее загружается. На самом деле страница не загрузится (вместе с выводом всех операторов «печати», написанных вами), пока процесс сбора данных не будет завершен полностью. Этот процесс может занять несколько минут, часов или вовсе не завершится никогда, в зависимости от того, как скрапер написан. Хотя скрапер, безусловно, выполняет свою работу, вам, возможно, будет более интересен вывод

результатов в режиме реального времени. Для этого вам потребуется настоящий сервер.

Запуск из облака

На заре развития компьютерных технологий программисты покупали или резервировали время на компьютерах, чтобы выполнить свой программный код. С появлением персональных компьютеров это стало ненужным, вы просто пишете и выполняете код на вашем собственном компьютере. Теперь аппетиты приложений опережают возможности процессора настолько, что программисты в очередной раз прибегают к платным сервисам.

Однако на этот раз пользователи платят за время работы не на одной физической машине, а за эквивалентные вычислительные ресурсы, часто распределяемые по нескольким машинам. Облачная структура этой системы предлагает вычислительные ресурсы, стоимость аренды которых определяется в соответствии с периодом пиковой нагрузки. Например, Amazon позволяет предлагать свою цену на свободные вычислительные сервисы (так называемые спотовые сервисы или спотовые инстансы), когда низкая цена важнее оперативности.

Кроме того, вычислительные сервисы становятся более специализированными, и их можно выбрать, исходя из потребностей вашего приложения, использовав такие настройки, как «большой объем памяти», «высокая скорость вычислений» и «большой объем физического пространства». Веб-скраперы обычно не используют много памяти, но для Вашего скрапера вы можете выбрать сервис, предлагающий большой объем физического пространства или высокую скорость вычисления. Если вы выполняете большой объем работ, связанный с обработкой естественного языка, распознаванием текста или нахождением кратчайшего пути между ссылками (как, например, в случае с «Шестью шагами Википедии»), выбирайте сервис с высокой скоростью вычислений. Если вы осуществляете скрапинг больших объемов данных, сохраняете файлы или выполняете крупномасштабный анализ данных, выбирайте сервис с большим объемом физического пространства.

Несмотря на то что вычислительные ресурсы облака ограничены лишь вашими финансовыми возможностями, на момент написания этой книги стоимость услуг вычислительных сервисов начиналась от 1,3 цента в час (для сервиса Amazon EC2), а самый дешевый сервис Google предлагает свои услуги за 4,5 цента в час с минимальным периодом использования 10 минут. Благодаря низким ценам аренда не-

большого вычислительного сервиса примерно сопоставима с покупкой своего собственного железа, за исключением того, что теперь вам не нужно нанимать айтишника для его обслуживания.

Конечно, пошаговые инструкции по настройке и запуску облачных вычислительных серверов выходят за рамки этой книги, но вы, скорее всего, придете к выводу, что эти пошаговые инструкции и не нужны. И Amazon и Google (не говоря уже о бесчисленных небольших компаниях) соперничают на рынке облачных вычислений, они сделали процесс аренды новых серверов настолько легким, что вам нужно лишь задать имя приложения и указать номер кредитной карты. Кроме того, на момент написания книги и Amazon, и Google предлагали бесплатно воспользоваться сервисами, чтобы привлечь новых клиентов.

Арендовав сервис, вы получаете IP-адрес, имя пользователя и публичный/секретный ключ, которые используются для подключения к сервису через протокол SSH. С этого момента все выглядит точно так же, как если бы вы работали с вашим собственным сервером, за исключением того, что вам, конечно, больше не нужно его обслуживать и запускать множество сложных инструментов мониторинга.

Дополнительные ресурсы

Много лет тому назад работа «в облаке» была в основном уделом тех, кто чувствовал в себе силы пробраться сквозь дебри технической документации и уже имел некоторый опыт администрирования серверов. Но на сегодняшний день облачные инструменты стали значительно проще в связи с ростом популярности и конкуренции среди провайдеров облачных вычислений.

Однако для создания крупномасштабных или более сложных скраперов вам, скорее всего, потребуется более подробное руководство по созданию платформы для сбора и хранения данных.

Google Compute Engine Марка Коэна, Кэтрин Херли и Пола Ньюсона – простой источник, посвященный совместному использованию Google Cloud Computing как с Python, так и с JavaScript. В нем рассматривается не только пользовательский интерфейс Google, но инструменты, работающие на базе командной строки и скриптов, которые можно использовать, чтобы сделать Ваше приложение более гибким.

Если вы предпочитаете работать с Amazon, книга Митча Гарнаата Python and AWS Cookbook является кратким, но чрезвычайно полезным

руководством, которое поможет вам начать работу с веб-сервисами Amazon и покажет вам, как создать и запустить масштабируемое приложение.

Заглянем в будущее

Web постоянно меняется. Технологии, которые передают на наш компьютер изображения, видео, текст и другие файлы данных, постоянно обновляются и изобретаются заново. Для того чтобы идти в ногу, технологии, используемые для сбора данных из Интернета, необходимо также менять.

В будущем из текста этой книги полностью исчезнет JavaScript как устаревшая и редко используемая технология, и вместо нее мы сосредоточимся на парсинге голограмм HTML8. Однако то, что точно не изменится, – так это образ мыслей и общий подход, необходимый для успешного скрапинга любого веб-сайта. Столкнувшись с тем или иным проектом по веб-скрапингу, сейчас и в отдаленном будущем, вы обязательно должны спросить себя:

- На какой вопрос я хочу ответить или какую проблему я хочу решить?
- Какие данные помогут мне достичь этого и где они находятся?
- В каком виде эти данные представлены на сайте? Могу ли я точно определить, какой фрагмент кода веб-сайта содержит эту информацию?
- Как я могу отделить нужные данные от ненужных и извлечь их?
- Какие операции по обработке или анализу нужно выполнить, чтобы использовать данные с максимальной пользой?
- Как я могу сделать этот процесс лучше, быстрее и надежнее?

Кроме того, вам нужно не только понимать, как применять инструменты, представленные в этой книге, по отдельности, но и как их использовать вместе для решения более серьезных проблем. Иногда данные можно легко извлечь и они хорошо отформатированы, что позволяет использовать простой скрапер. В других случаях вы должны хорошо подумать, прежде чем создавать скрапер.

Например, в главе 10 я объединила библиотеку Selenium для поиска Ajax-загружаемых изображений на Amazon и Tesseract для распознавания этих изображений. При решении задачи «Шесть шагов Википедии» я использовала регулярные выражения, чтобы написать краулер, который сохранял информацию о ссылках в базе данных,

а затем использовал алгоритм на основе графов, чтобы ответить на вопрос: «Каков кратчайший путь между Кевином Бэйконом и Эриком Айдлом»?

Редко встретишь проблему, которую невозможно решить, когда дело доходит до автоматизированного сбора данных в Интернете. Просто помните: Интернет – это один гигантский API с несколько скудным пользовательским интерфейсом.

Приложение A

Кратко о том, как работает Python

Согласно данным Ассоциации по вычислительной технике, Python является главным языком обучения в США, на уроках программирования в начальных классах он преподается чаще, чем BASIC, Java и даже C. В этом разделе рассказывается об установке, использовании и работе Python 3.x и программ для Python.

Установка и «Hello, World!»

Если вы используете OS X или практически любую версию Linux, у вас, скорее всего, Python уже установлен. Если у вас возникли сомнения, введите в командной строке:

```
$python --version
```

В этой книге мы будем использовать Python 3.x. Если у вас установлен Python 2.x, в Linux можно просто обновить версию, вызвав `apt-get`:

```
$sudo apt-get install python3
```

Обратите внимание, что это может повлечь за собой изменение способа выполнения кода Python: в командной строке нужно будет набирать `$python3 myScript.py`, а не `$python myScript.py`.

На момент написания книги в операционной системе OS X по умолчанию был установлен Python 2.7. Чтобы установить Python 3.x, скачайте инсталлятор с веб-сайта Python. Опять же, если вы одновременно используете Python 2.x и Python 3.x, вам, возможно, понадобится явно вызвать Python 3.x с помощью `$python3`.

Пользователи Windows, у которых Python еще не установлен, могут воспользоваться скомпилированными инсталляторами. Просто скачайте, откройте и установите. Кроме того, возможно, потребуется за-

дать системный путь к каталогу, в котором установлен Python, чтобы сообщить операционной системе его месторасположение, впрочем, инсталляторы снабжены довольно простыми инструкциями, как это сделать.

Для получения более полной информации об установке и обновлении версии Python в различных операционных системах посетите страницу загрузок сайта Python.

В отличие от Java, Python можно использовать в качестве скриптового языка, не создавая новых классов или функций. Создайте новый текстовый файл, напишите:

```
print("Hello, Internet!")
```

и сохраните его как hello.py. Если вы хотите написать чуть более развернуто, можно создать новую функцию и использовать ее для выполнения той же самой операции:

```
def hello():
    print("Hello, Internet!")
hello()
```

Здесь нужно отметить несколько вещей:

Python не использует точку с запятой для обозначения конца строки, а также не использует фигурные скобки, чтобы указать начало и окончание цикла или функции. Вместо этого он использует символы разрыва строк и табуляции. Начинаем строку с ключевого слова def, после которого следует имя функции, аргументы (в данном случае у нас (), что обозначает отсутствие аргументов), а также двоеточие, указывающее на то, что далее следует тело функции. После объявления функции каждая строка функции начинается с отступа, следующая строка без отступа обозначает конец тела функции.

Если поначалу идея о таком использовании пробельного символа (whitespace) кажется вам немного педантичной (или безумной), задумайтесь о ней. Исходя из своего опыта, скажу, что мой способ писать код на Python на самом деле улучшил читаемость кода, написанного на других языках. Только не забудьте поставить точку с запятой, если вы используете языки типа Java или C!

Python является слабо типизированным языком, это означает, что объявлять переменные и явно указывать их тип (строка, целое число, объект и т. д.) не нужно. Как и в других слабо типизированных языках, это может иногда вызывать проблемы с отладкой, но объявление переменной выполняется быстро:

```
greeting = "Hello, Internet!"
print(greeting)
```

Вот и все! Вы – питонист!

Разумеется, одним из замечательных свойств Python является тот факт, что он является настолько простым языком, что программисты других языков могут читать и интерпретировать его без особой предварительной подготовки, хотя это не совсем так. Этот аспект лучше всего проиллюстрировать на примере самого известного питоновского «пасхального яйца»:

```
import this
```

В ответ выводится:

```
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch[1].
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

[1] Возможно, эта строчка является отсылкой к голландскому ученому Эдсгеру Дейкстре, который в 1978 году сказал: «Я полагал, что это был твердый принцип разработки языка... Возможности различного представления программ, эквивалентных друг другу во всех отношениях, должны быть ограничены... В противном случае неоправданно возникают совершенно разные стили программирования, тем самым затрудняя удобство эксплуатации, читаемость и прочее» (*http://www.cs.utexas.edu/~EWD/transcriptions/EWD06xx/EWD660.html*). Либо эта фраза просто объясняется тем, что первоначальный разработчик языка Python Гвидо ван Россум – голландец по национальности. Впрочем, в этом вопросе нельзя быть полностью уверенным.

Дзен Питона (Тим Петерс)

Красивое лучше, чем уродливое.
Явное лучше, чем неявное.
Простое лучше, чем сложное.
Сложное лучше, чем запутанное.
Плоское лучше, чем вложенное.
Разреженное лучше, чем плотное.
Читаемость важна.
Особые случаи не настолько особые, чтобы нарушать правила.
Хотя практичность важнее безупречности.
Ошибки никогда не должны замалчиваться.
Если не замалчиваются явно.
Встретив двусмысленность, отбрось искушение угадать.
Должен существовать один — и желательно только один —
очевидный способ сделать это.
Хотя он поначалу может быть и не очевиден, если вы не голландец.
Сейчас лучше, чем никогда.
Хотя никогда зачастую лучше, чем прямо сейчас.
Если реализацию сложно объяснить, то идея плоха.
Если реализацию легко объяснить, то идея, возможно, хороша.
Пространства имён — отличная штука. Давайте делать их больше!

Приложение B

Кратко о том, как работает Интернет

Поскольку типы осуществляемых интернет-взаимодействий становятся все более сложными, термины и технологии, используемые для описания этих взаимодействий, также усложняются. Интернет, уже давно не использующийся как средство обмена сообщениями между учеными, теперь должен обрабатывать большие файлы, потоковое видео, защищенные банковские операции, покупки по кредитным картам, а также осуществлять передачу конфиденциальных корпоративных документов.

Однако, несмотря на все эти сложности, Интернет по своей сути является набором сообщений. Некоторые сообщения содержат запросы о предоставлении информации, некоторые содержат сообщения, предназначенные для удаленного получателя, некоторые содержат информацию о файле или инструкции для конкретного приложения на компьютере, которому эта информация адресована. Эти запросы отправляются клиентской машиной (настольным компьютером или мобильным устройством) на сервер, и наоборот. Кроме того, сами серверы могут обмениваться этими сообщениями между собой, чтобы собрать больше информации, запрошенной клиентом.

На рис. B.1 показано несколько распространенных типов интернет-взаимодействий: запрос IP-адреса по конкретному доменному имени, запрос веб-страницы и связанного с ней файла изображения к двум серверам и загрузка файла изображения.

Существует много различных протоколов и языков, которые управляют этими взаимодействиями между клиентами и серверами. Вы можете получить почту по протоколу SMTP, сделать телефонный звонок с помощью VOIP и загрузить файлы через FTP. Все эти прото-

колы задают различные поля заголовков, кодировки данных, адреса отправки/получения или имена, а также другие параметры. Для выполнения запросов, отправки и получения веб-данных используется HTTP (Hypertext Transfer Protocol , или Протокол обмена гипертекстовой информацией).

Рис. В.1 ❖ Несколько типов распространенных интернет-взаимодействий между клиентами и серверами

Подавляющее большинство скраперов в этой книге (и, вероятно, большинство скраперов, которые будут написаны вами) использует протокол HTTP для взаимодействия с удаленными веб-серверами. По этой причине важно рассказать о данном протоколе чуть подробнее.

HTTP-сообщение состоит из двух основных частей: полей заголовка и поля данных. Каждое поле заголовка содержит имя и значение. Возможные имена этих полей заранее определены стандартом HTTP. Например, у вас может быть поле заголовка типа:

```
Content-Type: application/json
```

указывающее на то, что данные в HTTP-пакете будет использовать формат JSON. Существуют более 60 возможных полей заголовка, которые можно использовать в HTTP-пакете, но в этой книге мы будем использовать лишь небольшую часть из них. В следующей таблице

приведены некоторые поля HTTP-заголовка, с которыми вы должны ознакомиться:

Имя	Описание	Пример
User-Agent	Строка, в которой указывается, с помощью какого браузера и операционной системы выполняется запрос	Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:28.0) Gecko/20100101 Firefox/28.0
Cookie	Переменные, используемые веб-приложением для хранения сессионных данных и другой информации	"__utma: 20549163.147923691.1 398729710.1398729710.1398858 679.2"
Status	Код, указывающий успешность или неуспешность выполнения запроса страницы	"200" (Okay), "404" (Not Found)

После того как пакет, доставляемый с помощью HTTP, достигает вашего браузера, содержимое пакета должно быть интерпретировано как веб-сайт. Структуру веб-сайтов определяет HTML (Hypertext Markup Language, или язык гипертекстовой разметки). Хотя HTML часто относят к языкам программирования, HTML – это язык *разметки* (*markup*). Он определяет структуру документа с помощью тегов, для того чтобы обозначить такие элементы, как заголовок, основной текст, боковая панель, нижний колонтитул и многое другое.

Все HTML-страницы (по крайней мере, все хорошо отформатированные HTML-страницы) имеют открывающий и закрывающий теги `<html></html>` с тегами `<head>` и `<body>` между ними. Остальные теги располагаются внутри тегов `<head>` и `<body>`, чтобы сформировать структуру и содержимое страницы:

```
<html>
<head>
<title>An Example Page</title>
</head>
<body>
<h1>An Example Page</h1>
<div class="body">
Some example content is here
</div>
</body>
</html>
```

В этом примере заголовок страницы (это текст, отображаемый на вкладке браузера) – это «An Example Page» с тем же названием в теге заголовка `<h1>`. Ниже расположен тег div («divider», или «раздели-

тель») класса «body», содержащий то, что может быть основной статьей или большим куском текста.

Анализ веб-сайтов для проведения быстрого скрапинга

Тот факт, что содержимое страницы дублируется в различных областях страницы, можно с пользой использовать в веб-скрапинге – возможно, информацию, расположенную в одной части страницы, будет собрать легче и надежнее, чем в другой.

Конечно, в вышеприведенном примере с «An Example Page», название довольно очевидно дублируется в двух отдельных областях страницы, но в Интернете распространены более сложные случаи. Например, вы можете собрать с корпоративного сайта информацию об именах сотрудников и обратить внимание на то, что имена отформатированы по-разному:

```
<span id="mary_smith">Dr. Mary Smith, CEO</span>
<span id="john_jones">President of Finance Mr. John E. Jones</span>
<span id="stacy_roberts">Stacy Roberts III, Marketing</span>
```

Разбивка содержимого этих тегов на пары слов «firstname, lastname» была бы трудной. Как вы поступите с именами посередине, званиями, должностями и прочим мусором в этой мешанине форматов? К счастью, похоже, что теги id хорошо отформатированы, и их выделение в Python становится тривиальной задачей разбиения строки с помощью символа подчеркивания.

CSS (Cascading Style Sheets, или каскадные таблицы стилей) используется наряду с HTML для настройки внешнего вида сайта. CSS задает такие параметры, как цвет, положение, размер и фон различных объектов, размещаемых на веб-сайте.

Совместное использование CSS и HTML, взятого из предыдущего примера, может выглядеть примерно так:

```
h1{
    color:'red';
    font-size:1.5em;
},
div.body{
    border:2px solid;
}
```

В результате мы создаем средний по размерам залоговок красного цвета с рамкой вокруг основного текста сайта.

К сожалению, более подробное освещение HTTP, HTML и CSS выходит за рамки этой книги. Однако, если вы незнакомы с этими темами, я рекомендую вам заглянуть на W3Schools, чтобы разобраться в незнакомых терминах или строках HTML/CSS кода, которые вы встретили в этой книге. Использование настройки браузера «исходный код страницы» также станет большим подспорьем при знакомстве с синтаксисом.

Приложение C

Правовые и этические аспекты веб-скрапинга

В 2010 году инженер-программист Пит Уорден написал веб-краулер для сбора данных с Facebook. Он собрал данные о примерно 200 миллионах пользователей Facebook – их имена, информацию о местоположении, друзьях и интересах. Конечно, Facebook заметил это и отправил ему предупредительное письмо с требованием прекратить подобные действия, которому он повиновался. Когда его спросили, почему он выполнил требование, он ответил: «В отличие от больших данных, адвокаты не так дешевы».

В этой главе мы рассмотрим законы США (и некоторых международные законы), которые имеют отношение к веб-скрапингу, а также узнаем, как быть с легальными и этическими аспектами данного вопроса.

Перед тем как читать этот раздел, примите во внимание очевидную вещь: я инженер-программист, а не юрист. Соответственно, все, что вы прочитаете здесь или в любой другой главе книги, не следует рассматривать как профессиональную юридическую консультацию или руководство к действию. И хотя я считаю, что могу компетентно обсуждать правовые и этические вопросы веб-скрапинга, вы должны проконсультироваться с юристом (а не с инженером-программистом), прежде чем браться за выполнение проектов по веб-скрапингу, которые с юридической точки зрения могут трактоваться неоднозначно.

Товарные знаки, авторские права, патенты, о боже!

Вот и пришло время для интеллектуальной собственности! Существуют три основных типа интеллектуальной собственности: товарные

знаки (обозначенные знаком ™ или ®), авторские права (вездесущий ©) и патенты (иногда сопровождаются текстом, в котором говорится о том, что изобретение защищено патентом, но часто вообще ничего не указывается).

Патенты используются для объявления права собственности лишь на изобретения. Вы не можете запатентовать сами изображения, текст и любую другую информацию. Хотя некоторые патенты, например патенты на программное обеспечение, менее осязаемы, чем то, что мы подразумеваем под «изобретениями», имейте в виду, что речь идет о *предмете* (*thing*) или методе, который запатентован, а не об информации, содержащейся в патенте. До тех пор, пока вы не строите здания по собранным чертежам или пока кто-то не запатентовал метод веб-скрапинга, вы вряд ли нарушаете патент, собирая данные в вебе.

Товарные знаки тоже вряд ли будут проблемой, но все-таки некоторые моменты нужно рассмотреть. Согласно Ведомству по патентам и товарным знакам США:

> ***Товарный знак*** *– это слово, фраза, символ и/или изображение, которое идентифицирует и помогает отличить ваши товары от товаров конкурентов. Знак обслуживания – это слово, фраза, символ и/или изображение, которое идентифицирует и помогает отличить ваши услуги от услуг конкурентов. Термин «товарный знак» часто используется для обозначения как товарных знаков, так и знаков обслуживания.*

Кроме традиционного брендинга слов/символов, который приходит на ум, когда мы вспоминаем о товарных знаках, товарными знаками могут быть и другие атрибуты. Например, речь может идти о форме контейнера (я про бутылки Coca-Cola) или даже цвете (в первую очередь розовый цвет стекловолоконной изоляции Pink Panther от Owens Corning).

В отличие от патентов, право собственности на товарный знак в значительной степени зависит от контекста, в котором он используется. Например, если я захочу опубликовать сообщение в блоге с изображением логотипа Coca-Cola, то вполне могу сделать это (если не подразумевается, что мой пост в блоге был оплачен или опубликован Coca-Cola). Если бы я захотела изготовить новый безалкогольный напиток с тем же логотипом Coca-Cola, изображенным на упаковке, это было бы явное незаконное использование товарного знака. Аналогично я могу упаковать мой новый безалкогольный напиток в розовый изолятор Pink Panther, но не могу использовать тот же цвет для создания домашних изоляционных материалов в коммерческих целях.

Авторское право

И торговые знаки, и патенты имеют что-то общее в том плане, что должны быть официально зарегистрированы, чтобы быть признанными. Вопреки распространенному мнению это не относится к материалам, защищенным авторским правом. Что делает изображения, текст, музыку и т. д. защищенными с точки зрения авторского права? Вовсе не предупреждение «all rights reserved» внизу страницы или статус «unpublished». Каждый создаваемый вами материал автоматически подпадает под действие закона об авторском праве, как только вы воплощаете его в жизнь.

Бернская конвенция по охране литературных и художественных произведений, названная в честь швейцарского города Берна, где она была впервые принята в 1886 году, является международным стандартом авторского права. Эта конвенция, по сути, гласит, что каждая страна, подписавшая ее, должна предоставить гражданам других стран-участниц те же авторские права, что и своим собственным гражданам. На практике это означает, что как гражданин США вы можете быть привлечены к ответственности в США за нарушение авторских прав на произведение, написанное кем-либо, скажем, во Франции (и наоборот).

Очевидно, что авторское право представляет проблему для веб-скраперов. Если я соберу контент с чьего-то блога и размещу его в своем собственном блоге, на меня с большой вероятностью могут подать в суд. К счастью, есть несколько способов защиты, которые я использую, чтобы обосновать легальность скрапинга блогов в зависимости от его задач.

Во-первых, защита авторских прав распространяется только на творческие произведения. Она не распространяется на статистические данные или факты. К счастью, большая часть из того, что ищут веб-скраперы, – это факты и статистические данные. Веб-скрапер, который собирает стихи со всего Интернета и размещает их на вашем собственном веб-сайте, может нарушать закон об авторском праве, однако веб-скрапер, который собирает информацию о частоте публикации стихов за определенный период, вполне легален. Поэзия «в сыром виде» – это творческое произведение. А вот среднее количество слов в стихах, опубликованных на сайте за месяц, творчеством уже не является.

Даже контент, который публикуется дословно (в отличие от агрегированного/статистического контента, полученного из собранных сырых данных), не может быть нарушением закона об авторском пра-

ве, если эти данные – цены, имена руководителей компаний или какая-либо другая фактическая информация.

Даже контент, защищенный авторским правом, можно непосредственно использовать в разумных пределах в соответствии с Законом об авторском праве в цифровую эпоху (Digital Millennium Copyright Act или DMCA). В Законе об авторском праве в цифровую эпоху изложены некоторые правила автоматизированной обработки материалов, защищенных авторским правом. Закон об авторском праве в цифровую эпоху – довольно объемный документ, с большим количеством конкретных правил, регулирующих защиту авторских прав на всех видах цифровых носителей, начиная от электронных книг и заканчивая телефонами. Однако есть три основных раздела, которые представляют особый интерес:

- согласно принципу защиты «безопасная гавань» («safe harbor»), если вы собрали материал из источника, который, по Вашему мнению, содержит лишь свободный контент, но какой-то пользователь заявил авторские права на него, вы освобождаетесь от ответственности, при условии, что удалите объект авторского права, получив уведомление о нарушении;
- вы не можете обходить меры безопасности (например, защиту паролем), чтобы собрать контент;
- вы можете работать с контентом в соответствии с принципом «добросовестного использования» («fair use»), который принимает во внимание процент использованного авторского материала и цель, в соответствии с которой этот материал используется.

Короче говоря, вы никогда не должны публично размещать материал, защищенный авторским правом, без разрешения автора или иного владельца авторского права. Если вы сохранили авторский материал в вашей собственной непубличной базе данных, к которой у вас свободный доступ, и используете для анализа, ради бога. Если вы опубликуете эту базу данных на вашем веб-сайте для просмотра или скачивания, это уже будет нарушением. Если вы проводите анализ этой базы данных и публикуете статистическую информацию о частоте слов, рейтинг авторов по творческой плодовитости или результаты еще какого-то метаанализа данных, на здоровье. Если вам нужно дополнить этот метаанализ несколькими избранными цитатами или выборками данных, чтобы обосновать свою точку зрения, это тоже здорово, но, возможно, нужно подробнее изучить пункт о «добросовестном использовании» Закона об авторском праве в цифровую эпоху, чтобы убедиться в законности действий.

Посягательство на движимое имущество

Посягательство на движимое имущество в корне отличается от того, что мы подразумеваем под «незаконным посягательством» в том плане, что речь здесь идет не о недвижимости или земельной собственности, а о движимом имуществе (например, сервере). Оно возникает в тот момент, когда ваше право собственности нарушается таким образом, что вы не можете получить доступ к ней или использовать ее.

В эпоху облачных вычислений возникает соблазн не думать о веб-серверах как реальных, материальных ресурсах. Тем не менее мало того, что серверы состоят из дорогих компонентов, их нужно где-то хранить, обслуживать, охлаждать и снабжать огромным количеством электроэнергии. По некоторым оценкам, 10% мирового потребления электроэнергии потребляется компьютерами (если электроника в вашем доме не убеждает вас, обратите внимание на огромные серверные фермы Google, которые нужно подключить к крупным электростанциям).

Хотя серверы являются дорогими ресурсами, они интересны с юридической точки зрения в том плане, что, как правило, веб-мастера *хотят*, чтобы люди пользовались их серверами (т. е. посещали их веб-сайты). Просто они не хотят, чтобы люди потребляли эти ресурсы *слишком много*. Зайти на веб-сайт с помощью браузера, это ради Бога, однако запускать полномасштабную DDoS-атаку на сайт, очевидно, недопустимо.

Существуют три критерия, согласно которым ваши действия по сбору данных могут быть расценены как посягательство на движимое имущество:

- *отсутствие согласия*: поскольку веб-серверы открыты для всех, они, как правило, также «дают согласие» и на веб-скрапинг. Однако пользовательские соглашения многих веб-сайтов прямо запрещают использование скраперов. Кроме того, любые предупредительные письма с требованием прекратить скрапинг отменяют это согласие;
- *фактический ущерб*: серверы являются дорогостоящими. Кроме расходов на обслуживание сервера, если ваш скрапер «уронит» или ограничит его работоспособность, это может быть расценено как причинение «вреда»;
- *намеренность действий*: если вы пишете код, вы отдаете себе отчет в том, что делаете!

Ваши действия должны подпадать под все эти три критерия, чтобы вам было предъявлено обвинение в незаконном посягательстве на

движимое имущество. Однако если вы нарушаете пользовательское соглашение без причинения фактического вреда, не думаю, что вы застрахованы от судебного иска. Вы вполне могли нарушить законодательство об авторском праве, Закон об авторском праве в цифровую эпоху, Закон о компьютерном мошенничестве и злоупотреблении (Computer Fraud and Abuse Act), о котором чуть позже, или один из многочисленных законов, касающихся веб-скрапинга.

Охлаждаем пыл ваших ботов

Раньше веб-серверы были гораздо мощнее персональных компьютеров. На самом деле частично под «сервером» подразумевался «большой компьютер». Теперь все поменялось. Например, мой персональный компьютер имеет процессор 3,5 ГГц и 8 ГБ оперативной памяти, тогда как средний по возможностям вычислительный сервис Amazon (на момент написания этой книги) имеет 4 ГБ оперативной памяти и процессор 3 ГГц.

С хорошим подключением к Интернету и приличными характеристиками даже один персональный компьютер может создать серьезную нагрузку на большинство сайтов, парализовав их работу или вообще «уронив». Если «неотложки» ждать неоткуда и единственное спасение – собрать все данные с веб-сайта Васи Пупкина максимум за две секунды, действительно нет причин «ронять» сайт.

Говорят, если смотреть на запущенного бота, то он никогда не завершит свою работу. В силу некоторых причин иногда лучше оставить краулер работать всю ночь:

- если в вашем распоряжении есть около восьми часов, даже в черепашьем темпе (две секунды на страницу) вы сможете просканировать более 14 000 страниц. Когда время не является первостепенной проблемой, у вас не возникает желания увеличить скорость ваших краулеров.
- если целевая аудитория совпадает с вашим месторасположением, то, вероятно, нагрузка на веб-сайт ночью будет намного ниже и ваш краулер не будет усугублять нагрузку на сайт в часы пик.
- вы можете поспать, вместо того чтобы постоянно проверять логи в поисках новой информации. Подумайте о том, как это волнительно – проснуться утром, получив новые данные!

Рассмотрим следующие сценарии:

- вы используете веб-краулер, который сканирует сайт Васи Пупкина, собирая все или только определенные данные;
- вы используете веб-краулер, который сканирует сотни небольших сайтов, собирая все или только определенные данные;
- вы используете веб-краулер, который сканирует очень большой сайт, например Википедию.

В первом случае лучше оставить бот работать медленно, на ночь.

Во втором случае лучше просканировать каждый сайт в циклическом режиме, а не сканировать их медленно, каждый раз по одному. В зависимости от количества сканируемых сайтов это означает, что вы можете собирать данные настолько быстро, насколько это позволяет ваше подключение к Интернету и ресурсы компьютера, однако нагрузка на каждый удаленный сервер должна быть разумной. Вы можете сделать это на программном уровне, либо использовав нескольких потоков (где каждый отдельный поток выпол-

няет краулинг одного сайта и приостанавливает свое собственное выполнение), либо с помощью списков Python, чтобы вести статистику по сайтам. В третьем случае нагрузка, которую могут оказать на сайт Википедии ваше подключение к Интернету и компьютер, вряд ли будет заметна или кого-то озаботит. Однако совсем другое дело, если вы будете использовать распределенную сеть машин. Будьте осторожны и по возможности всегда спрашивайте разрешения у представителя компании.

Закон о компьютерном мошенничестве и злоупотреблении

В начале 1980-х годов компьютеры стали перекочевывать из научных кругов в деловую среду. Поначалу вирусы и черви рассматривались скорее как неудобство (или даже как хобби для забавы) и только потом как серьезное уголовное преступление, которое может повлечь за собой фактические денежные убытки. В ответ на это в 1986 году был принят Закон о компьютерном мошенничестве и злоупотреблении.

Хотя вы можете подумать, что данный закон касается лишь типичных злонамеренных хакеров, пишущих вирусы, он также имеет серьезные последствия для веб-скраперов. Представьте скрапер, который сканирует Интернет, ищет формы ввода с простыми паролями или собирает государственные секреты, случайно размещенные в скрытом, но доступном месте. Все эти действия являются незаконными (и это правильно) в соответствии с Законом о компьютерном мошенничестве и злоупотреблении.

Закон определяет семь основных уголовных преступлений, которые можно кратко изложить так:

- несанкционированный доступ к компьютерам, принадлежащим правительству США, и похищение информации с этих компьютеров;
- несанкционированный доступ к компьютеру и похищение финансовой информации;
- несанкционированный доступ к компьютеру, принадлежащему правительству США, с целью его дальнейшего использования;
- умышленное проникновение на любой защищенный компьютер с целью мошенничества;
- умышленное проникновение на компьютер без разрешения с целью нанесения ущерба этому компьютеру;
- разглашение/передача паролей или авторизационных данных компьютеров, используемых правительством США, финансовыми организациями внутренней или внешней торговли;

- вымогательство денег или любых других материальных ценностей путем причинения ущерба или под угрозой причинения ущерба любому защищенному компьютеру.

Короче говоря, держитесь подальше от защищенных компьютеров, не пытайтесь получить доступ к компьютерам (в том числе к веб-серверам), если у вас нет прав на это, и особенно держитесь подальше от компьютеров правительства США и финансовых организаций.

robots.txt и Пользовательское соглашение

С юридической точки зрения, пользовательское соглашение веб-сайта и файлы *robots.txt* занимают интересное положение. Если веб-сайт является публичным, то право веб-мастера запретить или предоставить программному обеспечению доступ к сайту является дискуссионным. Выражение «здорово, если для просмотра этого сайта вы используете браузер, лишь бы не самостоятельно написанную программу» неоднозначно.

Большинство сайтов имеет ссылку на Пользовательское соглашение внизу каждой страницы. Пользовательское соглашение – это больше, чем просто правила для веб-скраперов и средств автоматизированного доступа. В нем часто содержится информация о том, какие сведения веб-сайт собирает, что он с ними делает, и, как правило, отказ от правовой ответственности, согласно которому услуги веб-сайта предоставляются без каких-либо явных или подразумеваемых гарантий.

Если вы занимаетесь поисковой оптимизацией (search engine optimization, или SEO) или технологиями поиска, вы, вероятно, слышали о файле *robots.txt*. Зайдя на любой крупный веб-сайт в поисках файла *robots.txt*, вы найдете его в корневом каталоге веб-сайта *http://website.com/robots.txt*.

Синтаксис, используемый файлами *robots.txt*, был разработан в 1994 году во время первоначального бума технологий веб-поиска. Примерно в это же время такие поисковые системы, как AltaVista и Dogpile, прочесав весь Интернет, начали всерьез конкурировать с простыми каталогами сайтов, один из таких каталогов составлялся компанией Yahoo! Увеличение популярности веб-поиска означало взрывной рост не только количества веб-краулеров, но и доступности информации, собираемой этими веб-краулерами, для обычного пользователя.

Хотя сейчас мы бы восприняли такую доступность информации как должное, некоторые веб-мастера были шокированы, когда информация, которую они прятали глубоко в файловой структуре своего

сайта, оказалась на первой странице результатов поиска, выдаемой основными поисковыми системами. В ответ на это был разработан синтаксис для файлов *robots.txt*, названный Стандартом исключений для роботов (Robots Exclusion Standard).

В отличие от пользовательского соглашения, в котором о веб-краулерах говорится в широком смысле на простом человеческом языке, файлы *robots.txt* можно легко обрабатывать и выполнять с помощью автоматизированных программ. Хотя они могут показаться идеальной системой, позволяющей решить проблему нежелательных ботов раз и навсегда, имейте в виду следующее:

- синтаксис, используемый в файлах *robots.txt*, не является официально принятым стандартом. Он является широко используемым и часто соблюдаемым соглашением, но никто не запрещает создать свою собственную версию файла *robots.txt* (за исключением того факта, что ни один бот не признает или не подчинится ему, пока тот не станет популярным). Как уже было сказано, этот синтаксис является широко распространенным соглашением преимущественно потому, что он относительно прост и компаниям нет никакой нужды придумать свой собственный стандарт или пытаться улучшить его;
- файл *robots.txt*, не является обязательным для выполнения всеми веб-краулерами. Он является всего лишь знаком, который говорит: «Пожалуйста, не заходите на эти разделы сайты». Существует большое количество доступных библиотек веб-скрапинга, которые выполняют *robots.txt* (хотя часто это просто установка по умолчанию, которую можно изменить). Кроме того, часто приходится устранять препятствия, обусловленные самим файлом *robots.txt* (в конце концов, вам нужно скачать его, разобраться в том, что можно сканировать, а что нет, и только после этого выполнять скрапинг), вместо того чтобы просто двигаться вперед и выполнять скрапинг нужной страницы.

Синтаксис Стандарта исключений для роботов довольно прост. Как и в Python (и многих других языках), комментарии начинаются с символа #, а заканчиваются символом новой строки. Комментарии можно использовать в любом месте файла.

Файл начинается со строки `User-agent:`, здесь задается название бота, к которому применяются правила. После строки `User-agent` следует набор правил, либо `Allow:`, либо `Disallow:`, в зависимости от того, разрешено боту сканировать данный раздел сайта или нет. Звездочка (*) означает групповой символ. Если в качестве значения `User-agent`

указан символ *, то права доступа распространяются на любых ботов, запросивших файл *robots.txt*.

Если ранее заданное правило противоречит последующему, приоритет имеет последнее правило. Например:

```
#Добро пожаловать в мой файл robots.txt!
User-agent: *
Disallow: *

User-agent: Googlebot
Allow: *
Disallow: /private
```

В данном случае мы запрещаем доступ всем ботам ко всем страницам сайта, за исключением Googlebot, которому разрешен доступ ко всем страницам сайта, за исключением каталога */private*.

Файл *robots.txt* Twitter содержит четкие инструкции для ботов Google, Yahoo! и Yandex (популярной поисковой системы в России), Microsoft и других ботов или поисковых систем, которые мы здесь еще не перечислили. Раздел для Google (идентичный разделам для всех остальных ботов) выглядит следующим образом:

```
#Google Search Engine Robot
User-agent: Googlebot
Allow: /?_escaped_fragment_

Allow: /?lang=
Allow: /hashtag/*?src=
Allow: /search?q=%23
Disallow: /search/realtime
Disallow: /search/users
Disallow: /search/*/grid

Disallow: /*?
Disallow: /*/followers
Disallow: /*/following
```

Обратите внимание на то, что Twitter ограничивает сбор данных в тех разделах сайта, в которых он для этих целей применяет API. Поскольку Twitter имеет хорошо отрегулированный API (а также платный API)[1], то в интересах компании пресекать всякие «доморощенные» API, которые собирают информацию, сканируя сайт Twitter независимо друг от друга.

---

[1] Речь идет о бесплатном Twitter Search API и платном Twitter Firehose API. – *Прим. пер.*

Несмотря на то что файл, указывающий путь Вашему краулеру, поначалу воспринимается как ограничение, на самом деле он может оказаться благом для развития веб-краулинга. Если файл *robots.txt* запрещает сканирование определенного раздела сайта, по сути, вебмастер говорит, что он не против краулинга остальных разделов сайта (в конце концов, если бы он был против, то ввел бы ограниченный доступ к ним, сразу указав это в файле *robots.txt*).

Например, раздел файла *robots.txt* Википедии, в котором задаются правила для веб-скраперов (в отличие от поисковых систем), на удивление либерален. Дошло до того, что он содержит удобочитаемый текст, чтобы поприветствовать ботов (то есть нас!), и блокирует доступ лишь к нескольким страницам, в частности к странице входа, странице поиска и странице «Случайная статья»:

```
#
# Friendly, low-speed bots are welcome viewing article pages, but not
# dynamically generated pages please.
#
# Inktomi's "Slurp" can read a minimum delay between hits; if your bot supports
# such a thing using the 'Crawl-delay' or another instruction, please let us
# know.
#
# There is a special exception for API mobileview to allow dynamic mobile web &
# app views to load section content.
# These views aren't HTTP-cached but use parser cache aggressively and don't
# expose special: pages etc.
#
User-agent: *
Allow: /w/api.php?action=mobileview&
Disallow: /w/
Disallow: /trap/
Disallow: /wiki/Especial:Search
Disallow: /wiki/Especial%3ASearch
Disallow: /wiki/Special:Collection
Disallow: /wiki/Spezial:Sammlung
Disallow: /wiki/Special:Random
Disallow: /wiki/Special%3ARandom
Disallow: /wiki/Special:Search
Disallow: /wiki/Special%3ASearch
Disallow: /wiki/Spesial:Search
Disallow: /wiki/Spesial%3ASearch
Disallow: /wiki/Spezial:Search
Disallow: /wiki/Spezial%3ASearch
Disallow: /wiki/Specjalna:Search
Disallow: /wiki/Specjalna%3ASearch
Disallow: /wiki/Speciaal:Search
```

```
Disallow: /wiki/Speciaal%3ASearch
Disallow: /wiki/Speciaal:Random
Disallow: /wiki/Speciaal%3ARandom
Disallow: /wiki/Speciel:Search
Disallow: /wiki/Speciel%3ASearch
Disallow: /wiki/Speciale:Search
Disallow: /wiki/Speciale%3ASearch
Disallow: /wiki/Istimewa:Search
Disallow: /wiki/Istimewa%3ASearch
Disallow: /wiki/Toiminnot:Search
Disallow: /wiki/Toiminnot%3ASearch
```

Писать веб-краулеры, которые выполняют инструкции файла *robots.txt*, или нет – решать вам, но я очень рекомендую выполнить эти инструкции, особенно если вы используете краулеры, которые без разбора сканируют веб.

Три нашумевших случая в практике веб-скрапинга

Поскольку веб-скрапинг – необъятное поле деятельности, существует огромное количество возможностей попасть в юридическую передрягу. В этом разделе мы рассмотрим три случая, которые касались определенной законодательной формулировки, обычно применяющейся к веб-скраперам, и расскажем, как она была использована в данном случае.

eBay против Bidder's Edge и посягательство на движимое имущество

В 1997 году рынок мягких игрушек был на подъеме, технологический сектор бурлил, и онлайн-аукционы были новинкой Интернета. Компания под названием Bidder's Edge придумала и разработала новый тип аукционного сайта. Вместо того чтобы заставить вас ходить по интернет-аукционам, сравнивая цены, он собирал данные со всех текущих интернет-аукционов по конкретному продукту и указывал вам сайт с самой низкой ценой.

Bidder's Edge реализовала это с помощью целой армии веб-скраперов, постоянно отправляя запросы на веб-серверы различных аукционных сайтов, чтобы получить информацию о цене и продукте. Из всех аукционных сайтов eBay был самым крупным, и Bidder's Edge совершала к серверам eBay по 100 000 запросов в день. Даже по сегод-

няшним меркам – это большой трафик. Согласно данным eBay, трафик от скраперов Bidder's Edge составлял 1,53% от общего объема интернет-трафика в тот период, и eBay, конечно, не был рад этому факту.

Компания eBay отправила предупредительное письмо Bidder's Edge, одновременно предложив приобрести лицензию на использование данных. Однако переговоры по лицензированию не увенчались успехом, и Bidder's Edge продолжала собирать данные с сайта eBay.

eBay попыталась заблокировать IP-адреса, которые использовала Bidder's Edge, занеся в черный список 169 различных IP-адресов, хотя Bidder's Edge смогла обойти эту проблему с помощью прокси-серверов (прокси-серверы – серверы, пересылающие запросы от имени другой машины, но использующие собственный IP-адрес прокси-сервера). Я уверена, вы можете себе представить, что это было удручающее и временное решение для обеих сторон – Bidder's Edge постоянно пыталась найти новые прокси-серверы и купить новые IP-адреса, когда прежние IP-адреса оказывались заблокированными, в то время как eBay была вынуждена вести большие списки доступа IP (сюда добавьте значительные вычислительные расходы на проверку IP-адресов в каждом пакете).

Наконец, в декабре 1999 года eBay подала в суд на Bidder's Edge, обвинив в посягательстве на движимое имущество.

Поскольку серверы eBay были реальными, материальными ресурсами компании и компании eBay не понравилось то, как Bidder's Edge использовала ее серверы, посягательство на движимое имущество казалось идеально подходящим законом для применения. На самом деле в наше время посягательство на движимое имущество неразрывно связано с судебными исками о незаконном веб-скрапинге и чаще всего воспринимается как закон, регулирующий сферу IT.

Судьи постановили, что eBay может выиграть дело о посягательстве на движимое имущество, доказав две вещи:

- Bidder's Edge не имела разрешения на использование ресурсов eBay;
- eBay понесла финансовые потери в результате действий Bidder's Edge.

Учитывая предупредительное письмо eBay наряду с логами, в которых содержалась информация об использовании серверов, и фактические расходы, связанные с обслуживанием серверами, это не составило труда для eBay. Конечно, ни одна судебная тяжба не заканчивается легко: были поданы встречные иски, потрачены деньги на большое количество адвокатов, и этот вопрос был урегулирован во внесудебном порядке за неизвестную сумму в марте 2001 года.

Означает ли это, что любое несанкционированное использование чужого сервера будет автоматически расценено как посягательство на движимое имущество? Необязательно. История с Bidder's Edge была особым случаем. Bidder's Edge использовала ресурсы eBay настолько интенсивно, что eBay была вынуждена приобрести дополнительные серверы, платить больше за электроэнергию и, возможно, нанять дополнительный персонал (хотя 1,53% может показаться незначительной цифрой, для крупной компании это может вылиться в значительную сумму).

В 2003 году Верховный суд Калифорнии вынес решение по другому делу, Intel против Хамиди, в котором бывший сотрудник Intel (Хамиди) отправлял сотрудникам Intel на серверы компании электронные письма, разоблачающие Intel. Суд постановил:

Претензия Intel отклоняется не потому, что электронная почта, передаваемая через Интернет, пользуется уникальным иммунитетом, а потому, что в Калифорнии посягательство на движимое имущество – в отличие от вышеупомянутых исковых оснований – не может быть установлено без предъявления доказательств вреда личной собственности истца или законным правам на нее.

По сути, Intel не удалось доказать, что шесть электронных писем, разосланных Хамиди всем сотрудникам (что интересно, каждый из них имел возможность отписаться от рассылки – по крайней мере, Хамиди был вежлив!), нанесли компании какой-либо финансовый убыток. Компания Intel не лишилась собственности или возможности ее использовать.

США против Орнхаймера и Закон о компьютерном мошенничестве и злоупотреблении

Если в Интернете размещена информация, которую человек может легко прочитать с помощью веб-браузера, то маловероятно, что он нарвется на федералов, считав ту же самую информацию в автоматическом режиме. Однако для достаточно любопытного человека обнаружение небольшой утечки безопасности является настолько простым делом, что маленькая утечка может очень быстро стать гораздо больше и гораздо опаснее, когда в дело вступят автоматизированные скраперы.

В 2010 году Эндрю Орнхаймер и Дэниэл Шпитлер заметили интересную особенность компьютеров iPad: при посещении веб-сайта AT&T происходило перенаправление на URL-адрес, содержащий уникальный ID компьютера IPad:

```
https://dcp2.att.com/OEPClient/openPage?ICCID=<idNumber>&IMEI=
```

Данная страница содержала форму входа с адресом электронной почты пользователя, ID которого был указан в URL-адресе. Это позволяло пользователям получить доступ к своим счетам, просто введя пароль.

Хотя существовало множество возможных идентификационных номеров, используя достаточное количество веб-скраперов, можно было перебрать эти номера и попутно собрать адреса электронной почты. Предоставив пользователям удобную функцию входа в систему, AT&T, по сути, опубликовал в Интернете электронные адреса своих клиентов.

Орнхаймер и Шпитлер создали скрапер, которая собрал 114 000 адресов электронной почты, в том числе электронные адреса знаменитостей, директоров компаний и правительственных чиновников. Затем Орнхаймер (но не Шпитлер) отправил список электронных адресов, а также информацию о том, как он был получен, в интернет-издание Gawker Media, которое и опубликовало эту историю (но не список) под заголовком «Apple's Worst Security Breach: 114,000 iPad Owners Exposed».

В июне 2011 года ФБР провело обыск в доме Орнхаймера в связи со сбором адресов электронной почты, хотя в конечном итоге они арестовали его по обвинению в торговле наркотиками. В ноябре 2012 года он был признан виновным в мошенничестве с использованием личных данных и сговоре с целью получения несанкционированного доступа к компьютеру, а через некоторое время его приговорили к 41 месяцу в федеральной тюрьме и обязали выплатить $73 000 в качестве компенсации.

Его случай привлек внимание адвоката по гражданским правам Орина Керра, который присоединился к его команде юристов и обжаловал дело в Апелляционном суде третьего округа. 11 апреля 2014 года (подобные судебные процессы могут быть весьма продолжительными), Апелляционный суд третьего округа согласился с жалобой, постановив:

Обвинение Орнхаймера по пункту 1 должно быть снято, потому что, согласно Закону о компьютерном мошенничестве и злоупотреблении, параграф 1030(a)(2)(C) раздела 18 Свода законов США, посещение общедоступного веб-сайта является санкционированным доступом. AT&T решил не использовать паролей или любых других защитных мер для обеспечения контроля за доступом к электронным адресам своих клиентов. Не имеет значения, что AT&T субъективно руководствовался стремлением обеспечить посторонним лицам беспрепятственный доступ к данным, или то, что Орнхай-

мер охарактеризовал этот доступ как «кражу». Компания сконфигурировала свои серверы так, чтобы сделать информацию доступной для всех, и таким образом разрешила широкой общественности просматривать информацию. Доступ к адресам электронной почты через публичный веб-сайт AT&T был разрешен в соответствии с Законом о компьютерном мошенничестве и злоупотреблении и поэтому не был преступлением.

Таким образом, здравый смысл возобладал в правовой системе, Орнхаймер был освобожден из тюрьмы в тот же день, и все стали жить долго и счастливо.

И хотя в итоге было вынесено решение о том, что Орнхаймер не нарушал Закона о компьютерном мошенничестве и злоупотреблении, его дом обыскивали агенты ФБР, он потратил много тысяч долларов на оплату юридических услуг, а также провел три года в залах суда и тюрьме. Применительно к теме веб-скрапинга: какие уроки мы можем извлечь из этого случая, чтобы избежать подобных ситуаций?

Если у вас нет знакомого адвоката, то скрапинг какой-либо конфиденциальной информации, будь то личные данные (в данном случае адреса электронной почты), коммерческая тайна или государственная тайна – это, вероятно, не совсем то, что вам нужно. Даже если эти данные находятся в открытом доступе, подумайте: «смог бы средний пользователь компьютера легко получить доступ к этой информации, если бы компания хотела показать эту информацию?», «это то, что компания разрешает смотреть пользователям?»

Я во многих случаях звонила в компании, чтобы сообщить об уязвимостях их веб-сайтов и веб-приложений. Этот способ творит чудеса: «Привет, я специалист по безопасности, который обнаружил потенциальную уязвимость на вашем сайте. Не могли бы вы соединить меня с кем-либо, кому я могу сообщить об этом и помочь решить эту проблему?» Помимо непосредственного удовлетворения от осознания себя хакерным гением («белой шляпой»), вы, возможно, получите бесплатные подписки, денежные вознаграждения и другие «вкусняшки»!

Кроме того, предоставление Орнхаймером информации интернет-изданию Gawker Media (перед тем как уведомить AT&T) и его самореклама, эксплуатирующая тему уязвимости, также сделали его особенно привлекательной мишенью для адвокатов AT&T.

Если вы обнаружили уязвимости на сайте, то лучшее, что вы можете сделать, – это предупредить владельцев сайта, а не СМИ. У вас может возникнуть соблазн написать сообщение в блоге и объявить об

этом всему миру, особенно если проблема не исправлена сразу. Однако вы должны помнить, что это зона ответственности компании, а не Ваша. Самое лучшее, что вы можете сделать, – это увести ваши скраперы (и, если это применимо, ваш проект) подальше от этого сайта!

Филд против Google: авторское право и robots.txt

Адвокат Блейк Филд подал иск против Google на том основании, что функция кэширования сайтов, реализуемая Google, нарушила закон об авторском праве, выводя кэшированную копию его книги после того, как он удалил книгу со своего сайта. Закон об авторском праве позволяет автору контролировать распространение своего произведения. Аргументом Филда был тот факт, что занесение книги в кэш (после того как он удалил ее со своего сайта) лишило его возможности контролировать распространение произведения.

Веб-кэширование Google

Когда веб-скраперы Google («Google bots») сканируют сайты, они делают копию сайта и размещают ее в Интернете. Любой желающий может получить доступ к кэшированной версии сайта, используя URL-формат:

```
http://webcache.googleusercontent.com/search?q=cache:http
://pythonscraping.com/
```

Если веб-сайт, который вы ищете или используте в ходе скрапинга, недоступен, проверьте наличие копии этого сайта, которую можно использовать для работы.

Зная о функции кэширования Google и не предприняв никаких действий, Филд не смог выиграть дело. В конце концов, он мог бы запретить ботам Google кэшировать его веб-сайт, просто добавив файл *robots.txt* с простыми инструкциями, указывающими, с каких страниц можно собирать информацию, а с каких – нет.

Что еще более важно, суд пришел к выводу, что положение «Безопасная гавань» Закона об авторском праве в цифровую эпоху позволило Google легально кэшировать и отображать сайты вроде сайта Филда: «[а] поставщик услуг не несет материальной ответственности... за нарушение авторских прав по причине промежуточного и временного хранения материалов в системе или сети, контролируемой или эксплуатируемой поставщиком услуг или от его имени».

Об авторе

Райан Митчелл – инженер-программист в компании «LinkeDrive» (Бостон), где она разрабатывает API и средства анализа данных. Она закончила Инженерный колледж имени Франклина Олина и является студентом-магистром Школы расширенного образования при Гарвардском университете. До прихода в компанию «LinkeDrive» Райан занималась созданием веб-скраперов и ботов в компании «Abine Inc.» и в данный момент выступает как консультант в проектах по веб-скрапингу, главным образом в финансах и розничной торговле.

Колофон

Животное, изображенное на обложке книги «Скрапинг веб-сайтов с помощью Python» – степной панголин (*Smutsia temminckii*). Панголин – млекопитающее, ведущее обособленный, ночной образ жизни, является родственником броненосцев, ленивцев и муравьедов. Они обитают в Южной и Восточной Африке. В Африке обитают еще три вида панголинов, и все они находятся под угрозой исчезновения.

Взрослые степные панголины могут достигать в среднем 30–98 см в длину и весят от 1,6 до 33 кг. Они напоминают броненосцев, покрытых защитными чешуйками, которые имеют либо темно-, светло-коричневый или оливковый цвет. У панголина, не достигшего зрелости, чешуйки имеют более розовый цвет. Когда им угрожает опасность, чешуйки на хвостах могут выступать в качестве средства нападения, поскольку они способны резать и ранить противников. Кроме того, панголин использует оборонительную стратегию, аналогичную скунсам, выделяя зловонный секрет из желез, расположенных у основания ануса. Это служит предупреждением для потенциальных противников, а также помогает панголину метить территорию. Живот панголина покрыт не чешуйками, а короткой шерстью.

Как и их родственники муравьеды, панголины питаются муравьями и термитами. Их невероятно длинные языки позволяют им найти еду в муравейниках и расщелинах. Язык у панголинов длиннее тела, во время сна они втягивают его в грудную полость.

Хотя панголины являются одиночными животными, взрослые особи живут в больших норах, которые располагаются глубоко под землей. Как правило, эти норы раньше принадлежали трубкозубам и бородавочникам, и панголины просто перебираются в заброшенное жилище. С помощью трех длинных изогнутых когтей на передних лапах панголины при необходимости могут без проблем выкопать свои собственные норы.

Многие животные, изображенные на обложках книг издательства O'Reilly, находятся под угрозой исчезновения, все они имеют важное значение для окружающего мира. Чтобы узнать больше о том, как вы можете помочь, зайдите на сайт *animals.oreilly.com*.

Изображение для обложки взято из книги *Ричарда Лидеккера «Royal Natural History»*. Шрифт текста – PetersburgC, шрифт заголовков – MagistralC и Arial, шрифт программного кода – Dalton Maag's Ubuntu Mono.

Предметный указатель

Книги издательства «ДМК Пресс» можно заказать
в торгово-издательском холдинге «Планета Альянс» наложенным платежом,
выслав открытку или письмо по почтовому адресу:
115487, г. Москва, 2-й Нагатинский пр-д, д. 6А.
При оформлении заказа следует указать адрес (полностью),
по которому должны быть высланы книги;
фамилию, имя и отчество получателя.
Желательно также указать свой телефон и электронный адрес.
Эти книги вы можете заказать и в интернет-магазине: **www.alians-kniga.ru**.
Оптовые закупки: тел. **(499) 782-38-89**.
Электронный адрес: **books@alians-kniga.ru**.

Райан Митчелл

Скрапинг веб-сайтов с помощью Python

Сбор данных из современного Интернета

Главный редактор *Мовчан Д. А.*
dmkpress@gmail.com
Научный редактор *Киселев А. Н.*
Переводчик *Груздев А. В.*
Корректор *Синяева Г. И.*
Верстка *Чаннова А. А.*
Дизайн обложки *Мовчан А. Г.*

Формат 60×90 1/16.
Гарнитура «Петербург». Печать офсетная.
Усл. печ. л. 24. Тираж 200 экз.

Веб-сайт издательства: www.дмк.рф