Посвящается AZ, NZ и TS: Twisted добился признания, и мы с нетерпением ждем следующего поколения разработчиков.

– Моше Задка (Moshe Zadka)

Содержание

Об авторах

Марк Уильямс (Mark Williams) работает в Twisted. В eBay и PayPal Марк Уильямс работал над высокопроизводительными веб-службами Python (более миллиарда запросов в день!), над обеспечением безопасности приложений и информации, а также переносом корпоративных Python-библиотек на Python.

Кори Бенфилд (Cory Benfield) – разработчик Python с открытым исходным кодом, активно участвует в сообществе Python HTTP. Входит в число основных разработчиков проектов Requests и urllib3 и является ведущим сопровождающим проекта Hyper – коллекции инструментов поддержки HTTP и HTTP/2 для Python, а также помогает в разработке Python Cryptographic Authority для PyOpenSSL.

Брайан Уорнер (Brian Warner) – инженер по безопасности и разработчик программного обеспечения, работавший в Mozilla на Firefox Sync, Add-On SDK и Persona. Один из основателей проекта Tahoe-LAFS – распределенной и защищенной файловой системы, разрабатывает средства безопасного хранения и связи.

Моше Задка (Moshe Zadka) с 1995 года является участником сообщества открытого исходного кода, свой первый вклад в Python внес в 1998 году, один из основателей открытого проекта Twisted. Любит рассказывать о Twisted и Python и выступать на конференциях. Регулярно ведет блоги.

Дастин Митчелл (Dustin Mitchell) внес свой вклад в Buildbot. Является членом команды TaskCluster в Mozilla. Также работает в командах Release Engineering, Release Operations и Infrastructure.

Кевин Сэмюэл (Kevin Samuel) начал заниматься разработкой и преподаванием еще во времена, когда появилась версия Python 2.4. Работал в Восточной Европе, Северной Америке, Азии и Западной Африке. Тесно сотрудничает с командой Crossbar.io и является активным членом французского сообщества Python.

О технических рецензентах

Пьер Тарди (Pierre Tardy) – специалист по непрерывной интеграции в Renault Software Labs, в настоящее время является ведущим коммитером в Buildbot.

Джулиан Берман (Julian Berman) – разработчик программного обеспечения с открытым исходным кодом из Нью-Йорка. Автор библиотеки jsonschema для Python, периодически вносит вклад в экосистему Twisted, активный участник сообщества Python.

Шон Шоджи (Shawn Shojaie) живет в районе Калифорнийского залива, где работает инженером-программистом. Работал в Intel, NetApp. Сейчас работает в SimpleLegal, где создает веб-приложения для юридических фирм. В будние дни занимается разработкой с использованием Django и PostgreSQL, а в выходные вносит вклад в проекты с открытым исходным кодом, такие как django-pylint, и время от времени пишет технические статьи. Больше можно узнать на сайте shawnshojaie.com.

Том Мост (Tom Most) – Twisted-коммитер с десятилетним опытом разработки веб-служб, клиентских библиотек и приложений командной строки с использованием Twisted. Инженер-программист в телекоммуникационной отрасли. Сопровождает Afkak, клиента Twisted Kafka. Его адрес в интернете – freecog.net, а связаться можно по адресу twm@freecog.net.

Благодарности

Благодарю свою жену Дженнифер Задка (Jennifer Zadka), без чьей поддержки я бы не справился.
Благодарю своих родителей Якова и Пнине Задка (Yaacov and Pnina Zadka), которые хорошо меня учили.
Благодарю своего советника Яэля Каршону (Yael Karshon) за то, что он научил меня писать.
Благодарю Махмуда Хашеми (Mahmoud Hashemi) за вдохновение и поддержку.
Благодарю Марка Уильямса (Mark Williams) за то, что всегда был рядом со мной.
Благодарю Глифа Лефковица (Glyph Lefkowitz), который познакомил меня с Python, учил программировать и вообще быть хорошим человеком.

– Моше Задка (Moshe Zadka)

Благодарю Махмуда Хашеми (Mahmoud Hashemi) и Давида Карапетяна (David Karapetyan) за их отзывы. Благодарю Энни (Annie) за то, что была терпелива, пока я писал.

– Марк Уильямс (Mark Williams)

Введение

Twisted недавно отпраздновал свой шестнадцатый день рождения. За время своего существования он превратился в мощную библиотеку. За этот период на основе Twisted было создано несколько интересных приложений и многие из нас узнали много нового о том, как правильно использовать Twisted, как думать о сетевом коде и как создавать программы, основанные на событиях.

После ознакомления с вводными материалами, которые есть на сайте Twisted, часто можно услышать: «И что делать дальше? Как я могу узнать больше о Twisted?» На этот вопрос мы обычно отвечали встречным вопросом: «А что вы хотите сделать с Twisted?» В этой книге мы покажем некоторые интересные приемы использования Twisted.

Авторы данной книги использовали Twisted для разных целей и усвоили разные уроки. Мы рады представить все эти уроки, чтобы сообщество их знало и могло воспользоваться.

Вперед!

От издательства

Отзывы и пожелания

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв прямо на нашем сайте www.dmkpress.com, зайдя на страницу книги, и оставить комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу dmkpress@gmail.com, при этом напишите название книги в теме письма.

Если есть тема, в которой вы квалифицированы, и вы заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу http://dmkpress.com/authors/publish_book/ или напишите в издательство по адресу dmkpress@gmail.com.

Скачивание исходного кода примеров

Скачать файлы с дополнительной информацией для книг издательства «ДМК Пресс» можно на сайте www.dmkpress.com на странице с описанием соответствующей книги.

Список опечаток

Хотя мы приняли все возможные меры для того, чтобы удостовериться в качестве наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг – возможно, ошибку в тексте или в коде, – мы будем очень благодарны, если вы сообщите нам о ней. Сделав это, вы избавите других читателей от расстройств и поможете нам улучшить последующие версии данной книги.

Если вы найдете какие-либо ошибки в коде, пожалуйста, сообщите о них главному редактору по адресу dmkpress@gmail.com, и мы исправим это в следующих тиражах.

Нарушение авторских прав

Пиратство в интернете по-прежнему остается насущной проблемой. Издательства «ДМК Пресс» и Apress очень серьезно относятся к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с незаконно выполненной копией любой нашей книги, пожалуйста, сообщите нам адрес копии или веб-сайта, чтобы мы могли применить санкции.

Пожалуйста, свяжитесь с нами по адресу dmkpress@gmail.com со ссылкой на подозрительные материалы.

Мы высоко ценим любую помощь по защите наших авторов, помогающую нам предоставлять вам качественные материалы.

Часть I

ОСНОВЫ

Глава 1

Введение в событийно-ориентированное программирование с помощью Twisted

Twisted – это мощная, хорошо протестированная, развитая параллельная сетевая библиотека и фреймворк. Как мы в этой книге увидим далее, многие организации и отдельные люди при создании своих проектов эффективно использовали этот фреймворк на протяжении более чем десятка лет.

В то же время Twisted большой, сложный и старый. Его лексикон изобилует странными названиями, такими как «реактор», «протокол», «конечная точка» и «отложенный». Эти понятия описывают философию и архитектуру, сбивающую с толку как новичков, так и людей с многолетним опытом работы с Python.

Сетевые приложения, создаваемые с помощью Twisted, основываются на двух основных принципах программирования: *событийного программирования* и *асинхронного программирования*. Рост популярности языка программирования JavaScript и включение в стандартную библиотеку Python пакета asyncio привели к тому, что оба принципа стали основой фреймворка. Но ни один из принципов не занимает настолько доминирующего положения в программировании на Python, чтобы простого знания языка было достаточно для их освоения. Эти сложные темы доступны, пожалуй, только программистам со средним или высоким уровнем подготовки.

В этой и следующей главах представлена мотивация, лежащая в основе событийно-управляемого и асинхронного программирования, а затем показано, как Twisted использует эти принципы. Здесь закладывается основа для последующих глав, в которых рассматриваются реальные программы Twisted.

Сначала мы познакомимся с идеей событийно-ориентированного программирования без привязки к Twisted. Затем, получив представление о том, что определяет событийно-управляемое программирование, мы познакомимся с программными абстракциями в Twisted, помогающими разработчикам писать четкие и эффективные событийные программы. Попутно мы будем делать остановки, чтобы познакомиться с некоторыми уникальными частями этих абстракций, например *интерфейсами*, и исследуем документацию с их описанием на веб-сайте Twisted.

К концу этой главы вы освоите терминологию Twisted: протоколы, транспорты, реакторы, потребители и производители. Данные понятия образуют основу событийно-ориентированного программирования с Twisted. Знание этой основы необходимо для разработки полезного программного обеспечения с Twisted.

Примечание о версиях Python

Twisted поддерживает Python 2 и 3, поэтому все примеры кода в этой главе могут работать как с Python 2, так и с Python 3. Python 3 – это будущее, но одной из сильных сторон Twisted является его богатая история реализации протоколов; по этой причине важно, чтобы вы умели писать код, способный выполняться под управлением Python 2, даже если прежде вы никогда не писали его.

Событийно-ориентированное программирование – что это?

Событие – это то, что заставляет управляемую событиями программу выполнить действие. Это широкое определение позволяет рассматривать многие программы как управляемые событиями. Рассмотрим, например, простую программу, которая в зависимости от ввода пользователя печатает либо `Hello`, либо `World`:

```
import sys
line = sys.stdin.readline().strip()
if line == "h":
    print("Hello")
else:
    print("World")
```

Появление строки в потоке стандартного ввода является событием. Наша программа приостанавливается на операторе `sys.stdin.readline()`, который просит операционную систему разрешить пользователю ввести законченную строку. Пока ввод не будет получен, программа дальше двигаться не будет. Когда операционная система прочитает очередной ввод и внутренние компоненты Python определят, что строка завершена, оператор `sys.stdin.`

readline() возобновит работу программы, вернув ей полученные данные. Возобновление работы – это событие, толкающее нашу программу вперед. Поэтому даже такую простую программу можно представить как управляемую событиями.

Многократные события

Программа, обрабатывающая единственное событие и завершающая работу, не получает никаких преимуществ от событийно-ориентированного подхода. Программы, в которых одновременно может происходить несколько событий, имеют более естественную организацию для их обработки. Примером такой программы может быть графический интерфейс пользователя: в любой момент пользователь может нажать кнопку, выбрать пункт из меню, прокрутить текст и т. д.

Вот версия нашей предыдущей программы с графическим интерфейсом Tkinter:

```
from six.moves import tkinter
from six.moves.tkinter import scrolledtext
class Application(tkinter.Frame):
    def __init__ (self, root):
        super(Application,self). __init__ (root)
        self.pack()
        self.helloButton = tkinter.Button(self,
                                    text="Say Hello",
                                    command=self.sayHello)
        self.worldButton = tkinter.Button(self,
                                     text="Say World",
                                     command=self.sayWorld)
        self.output = scrolledtext.ScrolledText(master=self)
        self.helloButton.pack(side="top")
        self.worldButton.pack(side="top")
        self.output.pack(side="top")
    def outputLine(self, text):
        self.output.insert(tkinter.INSERT, text+ '\n')
    def sayHello(self):
        self.outputLine("Hello")
    def sayWorld(self):
        self.outputLine("World")

Application(tkinter.Tk()).mainloop()
```

Эта версия нашей программы представляет пользователю две кнопки, каждая из которых может генерировать независимое событие (click) и отличается от предыдущей программы тем, что в предыдущей программе sys.stdin.readline может генерировать только одно событие: «готовность строки».

С этими событиями мы связали *обработчики событий*, которые вызываются нажатием любой кнопки. Кнопки Tkinter имеют свойство command со ссылкой

на обработчик, и вызывают этот обработчик при нажатии. Так, когда нажимается кнопка **Say Hello**, она генерирует событие, вынуждающее программу вызвать функцию `Application.sayHello`, которая, в свою очередь, выводит слово Hello в текстовое окно с прокруткой. То же происходит при нажатии кнопки **Say World**, которая вызывает функцию `Application.sayWorld` (см. рис. 1.1).

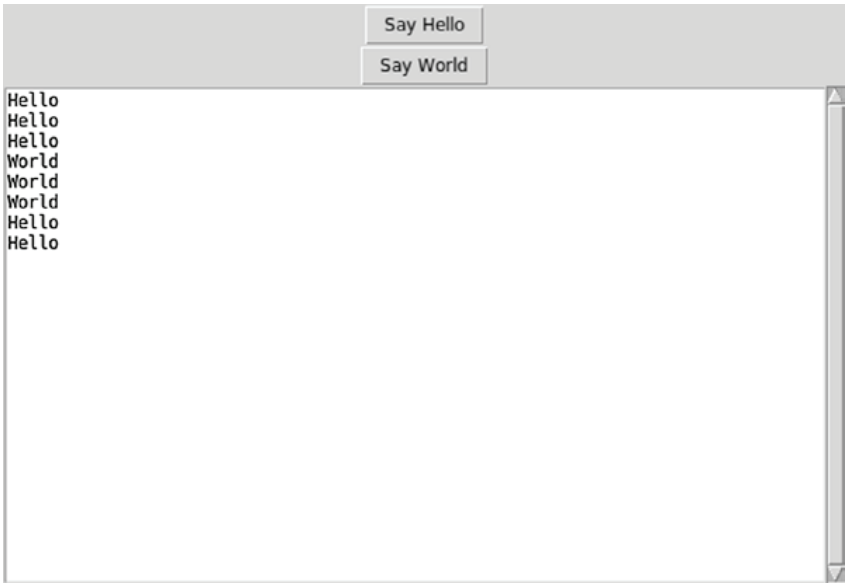

Рис. 1.1 ❖ Приложение Tkinter GUI
после нескольких нажатий кнопок **Say Hello** и **Say World**

Метод `tkinter.Frame.mainloop`, унаследованный нашим классом `Application`, ждет, пока связанная с ним кнопка сгенерирует событие, и запускает соответствующий обработчик. После выполнения каждого обработчика `tkinter.Frame.mainloop` переходит к ожиданию новых событий. Цикл ожидания событий и их передачи в соответствующие обработчики типичен для управляемых событиями программ и известен как *цикл событий*.

Вот основные понятия, лежащие в основе событийно-ориентированного программирования.

1. *События* показывают, что произошло то, на что должна реагировать программа. В обоих наших примерах события естественно соответствуют нажатиям кнопок, но, как мы увидим, они могут представлять все, что заставляет нашу программу выполнять какое-либо действие.
2. *Обработчики событий* определяют реакцию программы на события. Иногда обработчик события имеет вид простой инструкции, как вызов `sys.stdin.readline` в нашем примере. Но чаще всего он представлен функцией или методом, как в нашем примере `tkinter`.

3. *Цикл событий* ждет появления события и вызывает соответствующий обработчик. Не все управляемые событиями программы имеют цикл событий; в нашем первом примере `sys.stdin.readline` цикла обработки событий нет, потому что здесь происходит только одно событие. Однако большинство программ напоминают наш пример `tkinter` тем, что перед окончательным завершением обрабатывают много событий. Такие программы используют цикл событий.

Мультиплексирование и демультиплексирование

Способ ожидания появления событий в цикле существенно влияет на написание управляемых событиями программ, поэтому рассмотрим его более подробно. Вернемся к нашему примеру `tkinter` с двумя кнопками. Цикл событий внутри `mainloop` должен ждать, пока пользователь не нажмет одну из двух кнопок.

Простая реализация цикла может выглядеть следующим образом:

```
def mainloop(self):
    while self.running:
        ready = [button for button in self.buttons if button.hasEvent()]
        if ready:
            self.dispatchButtonEventHandlers(ready)
```

Цикл `mainloop` в ожидании нового события постоянно *опрашивает* каждую кнопку и запускает обработчики только для имеющих готовое событие. Когда событий нет, программа никаких действий не выполняет, так как никаких действий, требующих ответа, предпринято не было. В течение периодов бездействия управляемая событиями программа свое выполнение должна приостанавливать.

Цикл `while` в нашем примере `mainloop` приостанавливает работу программы до тех пор, пока не будет нажата одна из кнопок и не понадобится вызвать функцию `sayHello` или `sayWorld`. Если пользователь не сможет щелкать мышью со сверхъестественной скоростью, большая часть времени цикла будет тратиться на проверку кнопок, которые не были нажаты. Это называется *активным ожиданием*.

Активное ожидание, подобное этому, приостанавливает общее выполнение программы до тех пор, пока один из ее источников событий не сообщит о том, что событие произошло. Поэтому активного ожидания достаточно для приостановки цикла событий.

Внутренний генератор списков, управляющий активным ожиданием, задает важный вопрос: что-нибудь произошло? Ответ помещается в переменную `ready` и имеет вид списка со всеми кнопками, которые были нажаты. Истинность переменной `ready` определяет ответ на вопрос: если переменная `ready` ничего не содержит (имеет ложное значение), значит, кнопки не нажи-

мались и ничего не произошло. Если переменная имеет истинное значение, следовательно, событие произошло и по меньшей мере одна из кнопок была нажата.

Генератор списков, формирующий значение для `ready`, объединяет множество событий в одном месте. Этот процесс известен как *мультиплексирование*. Обратный процесс разделения списка на отдельные события называется *демультиплексированием*. Генератор списка объединяет (мультиплексирует) все наши кнопки в переменной `ready`, тогда как метод `dispatchButtonEventHandlers` демультиплексирует (перенаправляет сигнал с одного из информационных входов на один из информационных выходов) их, вызывая обработчик каждого события отдельно.

Теперь мы можем уточнить наше понимание циклов событий, точно описав, как происходит ожидание событий.

- *Цикл событий* ожидает появления событий, мультиплексируя их источники в один список. Если получился непустой список, цикл событий демультиплексирует его и для каждого события вызывает соответствующий обработчик.

Наш мультиплексор `mainloop` тратит бóльшую часть своего времени на опрос кнопок, которые не были нажаты. Но не все мультиплексоры настолько неэффективны. В `tkinter.Frame.mainloop` используется аналогичный мультиплексор, который опрашивает все виджеты, если операционная система не предлагает более эффективного способа. Мультиплексор `mainloop`, чтобы повысить свою эффективность, использует тот факт, что компьютеры могут проверять виджеты GUI быстрее, чем с ними может взаимодействовать человек, и вставляет вызов `sleep`, приостанавливающий программу на несколько миллисекунд. Это позволяет программе часть времени в цикле активного ожидания вообще не выполнять никаких операций и за счет небольшой задержки экономить процессорное время и электроэнергию.

Хоть Twisted может интегрироваться с графическими пользовательскими интерфейсами и фактически имеет специальную поддержку `tkinter`, в его основе лежит сетевой движок. В сети основными источниками событий являются *сокеты*, а не кнопки, и операционные системы предлагают эффективные механизмы для мультиплексирования событий сокетов. Цикл событий в Twisted использует эти механизмы. Чтобы понять подход Twisted к событийному программированию, нужно разобраться, как взаимодействуют сокеты с этими механизмами мультиплексирования.

Мультиплексор select

История, аналоги и назначение

Мультиплексор `select` поддерживается большинством операционных систем. Свое имя «select» (выбор) этот мультиплексор получил из-за своей способно-