Foundation Dynamic Web Pages with Python

Create Dynamic Web Pages with Django and Flask

David Ashley

Apress®

Foundation Dynamic Web Pages with Python

Create Dynamic Web Pages with Django and Flask

David Ashley

Apress®

__Foundation Dynamic Web Pages with Python__

David Ashley
Austin, TX, USA

ISBN-13 (pbk): 978-1-4842-6338-9 ISBN-13 (electronic): 978-1-4842-6339-6
https://doi.org/10.1007/978-1-4842-6339-6

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Celestin Suresh John
Development Editor: James Markham
Coordinating Editor: Divya Modi

Cover designed by eStudioCalamar

Cover image designed by Pixabay

Distributed to the book trade worldwide by Springer Science+Business Media New York, 1 New York Plaza, Suite 4600, New York, NY 10004-1562, USA. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail booktranslations@springernature.com; for reprint, paperback, or audio rights, please e-mail bookpermissions@springernature.com.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/978-1-4842-6338-9. For more detailed information, please visit www.apress.com/source-code.

Printed on acid-free paper

This book is dedicated to Debbie and Jim. A brother never had a better sister and brother-in-law.

Table of Contents

About the Author

David Ashley is a technical writer for SkillSoft where he specializes in open source, particularly Linux. As a member of the Linux Fedora documentation team he recently led the Libvirt project documentation and wrote the Python programs included with it. He has developed in 20 different programming languages during his 30 years as a software developer and IT consultant, including more than 18 years at IBM and 12 years with American Airlines.

About the Technical Reviewer

Akshay Saini is Mumbai-based tech book reviewer with several years' experience in IT as a software developer specializing in Python.

Recently he has developed new product solutions for the media and entertainment industry for clients such as Olympic Broadcasting Services (OBS), ViacomCBS, and Sportcast.

Reviewing and writing a tech book has always been on his bucket list, and with this book it became a reality.

He is a tech lover and spends much time developing software to build a better future for society.

Acknowledgments

I would like to acknowledge the people who contributed their time and efforts to this book. There are just too many to list here, but I would especially like to acknowledge all the people at Apress who helped put this book together.

Introduction

This book compares some of the best-known dynamic HTML page creation systems. It includes some older systems such as CGI and SSH as well as newer systems such as Flask and Django. Each system is examined and compared with the other systems to discover each of their strengths and weaknesses.

This should give you a basis for choosing the correct system for your dynamic HTML page needs. For each system, I will provide example programs so that if you are not experienced with a system, you can get a taste of what building an application with it is like.

Preface: Document Conventions

This book uses several conventions to highlight certain words and phrases and draw attention to specific pieces of information. The convention used depends on the type of information displayed.

Computer Commands

Computer commands are usually presented in a bold font such as in the following example:

`The Unix command` **`ls`** `run from the command shell is used to present a list of files and directories.`

Filenames

Filenames are usually presented in monospaced text such as in the following example:

To see the contents of the file `report.txt` use the command `cat report.txt`.

Programming Language Elements and Literals

Programming language elements include such things variable names, literals, constants, symbols, tokens, functions, class names, and other objects.

Literal data is taken directly from a computer screen or a computer language literal value and is usually presented in monospaced text such as within the following example:

The following line is the output from running **`ls`**:

```
en-US Makefile publican.cfg
```

Computer Output and Source Code

Computer output data is taken directly from a computer screen and usually presented in monospaced text such as in the following example. This information is usually set off from the rest of the text.

```
books        Desktop   documentation  drafts  mss    photos   stuff  svn
books_tests  Desktop1  downloads      images  notes  scripts  svgs
```

Source-code listings are also set off from the rest of the text, as shown in Listing P-1.

Listing P-1. This Is a Source Code Listing

```
from __future__ import print_function
import sys
import libvirt

conn = libvirt.open('qemu:///system')
if conn == None:
    print('Failed to open connection to qemu:///system', \
          file=sys.stderr)
    exit(1)
conn.close()
exit(0)
```

Notes and Warnings

Finally, we use three visual styles to draw attention to information that might otherwise be overlooked.

Note Notes are tips, shortcuts, or alternative approaches to the task at hand. Ignoring a note should have no negative consequences, but you might miss out on a trick that makes your life easier.

Important Important boxes detail things that are easily missed: configuration changes that apply only to the current session, or services that need restarting before an update will apply. Ignoring a box labeled "Important" will not cause data loss but may cause irritation and frustration.

Warning Warnings should not be ignored. Ignoring warnings will most likely cause data loss.

CHAPTER 1

Introduction to Web Servers

This chapter introduces web servers, the services they provide, and how they work. This information is essential to web developers so they can make proper use of their web server and provide the best web experience to their users.

All web servers use the same building blocks to serve up web pages to the user. While we could look at all the available web servers, it really is not necessary since they all are designed around the same building blocks. Instead, we will concentrate on the Apache web server since it is the most popular. All the other web servers use the same building blocks and design principles as the Apache server.

Glossary of Terms

When delving into technical information, it is important that you understand the terminology used. For that reason, please review the following web server terms:

- *Common Gateway Interface (CGI)*: This describes a process that serves up a dynamic web page. The web page is built by a program provided by the web server

D. Ashley, *Foundation Dynamic Web Pages with Python*,
https://doi.org/10.1007/978-1-4842-6339-6_1

administrator. A common set of information is available to the program via the program's environment.

- *Hypertext Transport Protocol (HTTP)*: This is a set of rules used to describe a request by the user to the web server and the returned information. The request and the reply must follow strict rules for the request to be understood by the server and for the reply to be understood by the user's browser.
- *Hypertext Markup Language (HTML)*: This code is used to build a web page that is displayed by a browser, and the Apache web server is used to serve the web page to users (clients). There are several versions of this code, but we will be using the latest version (5.0) in this book.
- *Cascading Style Sheets (CSS)*: These sheets define the styles to be used by one or more web pages. These styles are used to define fonts, colors, and the size of a section of text within a defined area of the HTML page.

These terms should give you a good starting point for discussing how a web server works. All of these terms will receive wider attention and definition throughout this book.

The Apache Web Server

The Apache web server (today this is known formally as the HTTP Server) dates back to the mid-1990s when it started gaining widespread use. The web server is a project of the Apache Software Foundation, which manages several projects. There are currently more than 200 million lines of code managed by the foundation for the Apache web server. The current release as of this writing is 2.4.41.

Starting with Apache version 2.0, Apache uses a hook architecture to define new functionality via modules. We will study this in a subsequent chapter.

When you first look at hooks, they will seem a little complicated, but in reality, they are not since most of the time you are only modifying Apache a little. This will reduce the code you need to write to implement a hook to a minimum.

The Apache web server uses a config file to define everything the server needs to know about all the hooks you want to include in Apache. It also defines the main server and any virtual servers you want to include. In addition, it defines the name of the server, the home directory for the server, the CGI directory to be used, any aliases needed by the server, the server name, any specific handlers used by that server, the port to be used by the server, the error log to be used, and several other factors.

Once configuration is complete, the Apache server is now ready to supply files to a client browser. This is called the *request-response cycle*. For each request sent by the browser to the server, the request must travel through the request-response cycle to produce a response that is sent back to the browser. While this looks simple on the surface, the request-response cycle is both powerful and flexible. It can allow programs you create, called *modules*, to modify both the request and the response in many flexible ways. A module can also create the response from scratch and can include inputs from resources outside of Apache, such as a database or other external data repository.

Figure 1-1 shows the request-response loop of Apache plus the startup and shutdown phases of the server.

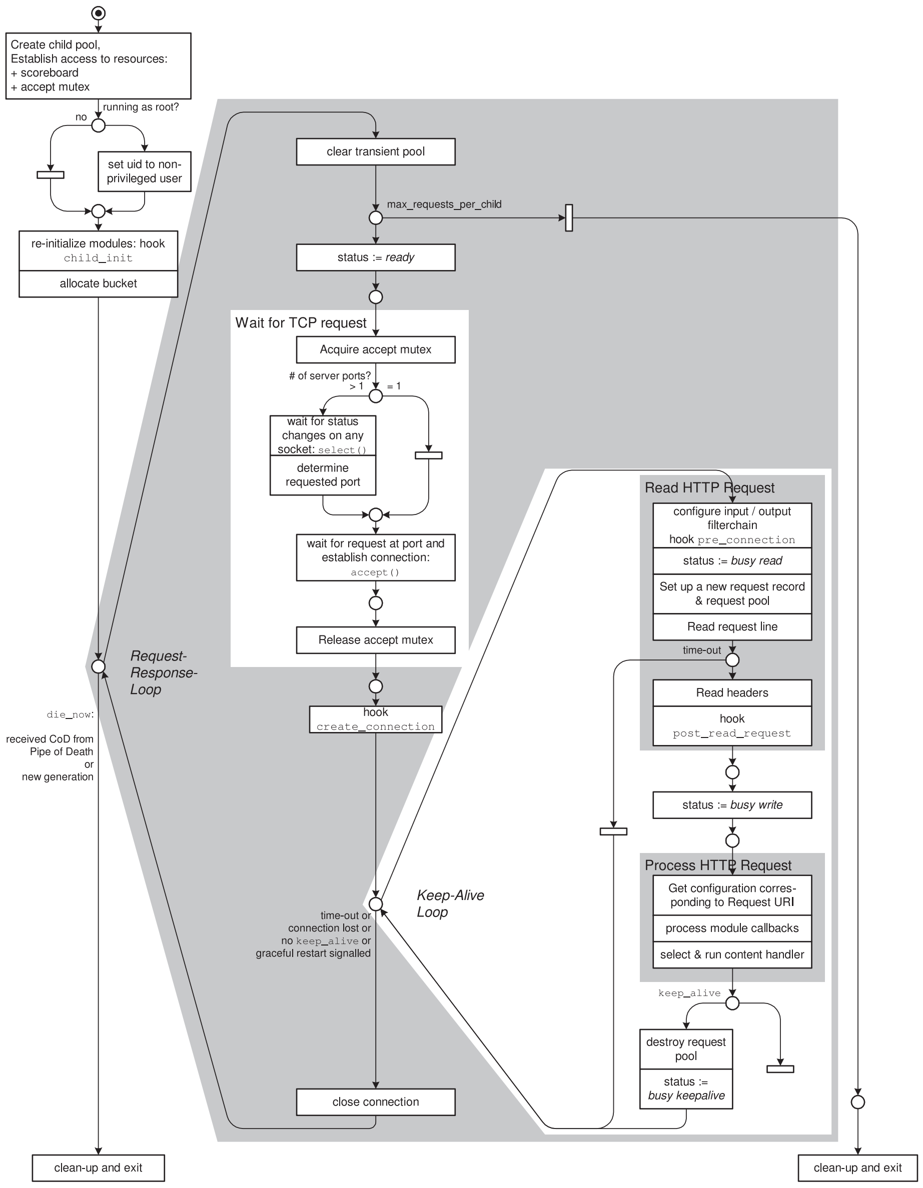

Figure 1-1. *The Apache request-response loop*

Modules not only can be used in the request-response cycle but in other portions of Apache such as during configuration, shutdown/cleanup, processing security requests, and other valuable functions. As you can see, modules allow flexible and powerful methods to be created by the server administrator to help with providing a great experience for their users.

Modules are not the only way to create dynamic web pages. Another way is by invoking available Apache services that can call an external program to create the page. The CGI process is usually invoked to supply this service, but there are other ways as well. Each of these ways will be examined in this book. It will be up to you to decide the best methodology for use in your environment.

The shaded request/response loop can have several forms. One such form is as a loop inside one of several processes under Apache. Another is running the loop as a thread inside a single process under Apache. All of these forms are designed to make the most efficient process of responding to a request that an operating system may provide.

The Keep-Alive loop is for HTTP 2.0 requests if supported by the web server. It allows the connection to stay open to the client until all requests have been processed. The loop here describes how a single request is processed by Apache. If the web server is not running HTTP 2.0 requests, then each request/response will close the connection once the response has been sent.

Nginx Web Server

The Nginx server was designed as a low-cost (in terms of system requirements) alternative to the Apache server. Probably the biggest difference between Nginx and Apache is that Nginx has an asynchronous event-driven architecture rather than using multiple threads to process each request. While this can provide predictable performance under high loads, it does come with some downsides. For instance, a request can end up waiting in the queue longer than the request attempt will survive on the network; i.e., the requester can give up before the request is ever processed if there are too few processing routines. While this problem is not exclusive to this server, it does still exist.

Recently the Nginx server has become popular within the community because of its smaller footprint and flexible design. However, since many of the principles that we will use to describe the Apache server also apply to the Nginx server, I will not delve deeply into Nginx and will discuss it only when differences between the two servers are important, especially in regard to dynamic web page design.

Apache Tomcat Server

The Apache Tomcat server is written in Java, which makes it difficult to compare to the more standard web servers. While some principles of dynamic web page design are similar, there are many differences. Therefore, and because it's less commonly used than Apache and Nginx, I will not attempt to cover it in this book.

Configuring the Apache Web Server

The Apache web server has a single main configuration file and a number of optional configuration files. The main file is named `httpd.conf`, and it controls which optional files are loaded as well as the location where they can be found. It also specifies the global features used by the server.

Listing 1-1 shows an unedited version of the `httpd.conf` file. Following the listing, I will describe the sections that need to be modified to give you a usable configuration file.

Listing 1-1. The Unedited httpd.conf File

```
#
# This is the main Apache HTTP server configuration file.
  It contains the
# configuration directives that give the server its
  instructions.
# See <URL:http://httpd.apache.org/docs/2.4/> for detailed
  information.
# In particular, see
# <URL:http://httpd.apache.org/docs/2.4/mod/directives.html>
# for a discussion of each configuration directive.
#
# See the httpd.conf(5) man page for more information on this
  configuration,
# and httpd.service(8) on using and configuring the httpd service.
#
# Do NOT simply read the instructions in here without
  understanding
# what they do.  They're here only as hints or reminders.
  If you are unsure
# consult the online docs. You have been warned.
#
# Configuration and logfile names: If the filenames you specify
  for many
# of the server's control files begin with "/" (or "drive:/"
  for Win32), the
# server will use that explicit path.  If the filenames do
  *not* begin
# with "/", the value of ServerRoot is prepended -- so
  'log/access_log'
```

```
# with ServerRoot set to '/www' will be interpreted by the
# server as '/www/log/access_log', where as '/log/access_log'
  will be
# interpreted as '/log/access_log'.

#
# ServerRoot: The top of the directory tree under which the
  server's
# configuration, error, and log files are kept.
#
# Do not add a slash at the end of the directory path.  If you point
# ServerRoot at a non-local disk, be sure to specify a local
  disk on the
# Mutex directive, if file-based mutexes are used.  If you wish
  to share the
# same ServerRoot for multiple httpd daemons, you will need to
  change at
# least PidFile.
#
ServerRoot "/etc/httpd"

#
# Listen: Allows you to bind Apache to specific IP addresses
  and/or
# ports, instead of the default. See also the <VirtualHost>
# directive.
#
# Change this to Listen on specific IP addresses as shown below to
# prevent Apache from glomming onto all bound IP addresses.
#
#Listen 12.34.56.78:80
Listen 80
```