

四种主流的Python Web开发框架详解



Python 高效开发实战

Django Tornado  
Flask Twisted

刘长龙 著



中国工信出版集团



电子工业出版社  
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY  
http://www.phei.com.cn

## 内 容 简 介

也许你听说过全栈工程师，他们善于设计系统架构，精通数据库建模、通用网络协议、后端并发处理、前端界面设计，在学术研究或工程项目上能独当一面。通过对 Python 及其周边 Web 框架的学习和实践，你就可以成为这样的全能型人才。

本书分为 3 部分：第 1 部分是基础篇，带领初学者实践 Python 开发环境和掌握基本语法，同时对网络协议、Web 客户端技术、数据库建模编程等网络编程基础深入浅出地进行学习；第 2 部分是框架篇，学习当前最流行的 Python Web 框架，即 Django、Tornado、Flask 和 Twisted，达到对各种 Python 网络技术融会贯通的目的；第 3 部分是实战篇，分别对 4 种框架进行项目实践，利用其各自的特点开发适用于不同场景的网络程序。

本书内容精练、重点突出、实例丰富、讲解通俗，是广大网络应用设计和开发人员不可多得的一本参考书，同时非常适合大中专院校师生学习和阅读，也可作为高等院校计算机及相关培训机构的教材。

未经许可，不得以任何方式复制或抄袭本书之部分或全部内容。  
版权所有，侵权必究。

## 图书在版编目（CIP）数据

Python 高效开发实战：Django、Tornado、Flask、Twisted / 刘长龙著. —北京：电子工业出版社，2016.10  
ISBN 978-7-121-30010-3

I. ①P… II. ①刘… III. ①软件工具—程序设计 IV. ①TP311.561

中国版本图书馆 CIP 数据核字（2016）第 236342 号

策划编辑：董 英

责任编辑：徐津平

印 刷：中国电影出版社印刷厂

装 订：三河市良远印务有限公司

出版发行：电子工业出版社

北京市海淀区万寿路 173 信箱 邮编：100036

开 本：787×980 1/16 印张：32.25 字数：680 千字

版 次：2016 年 10 月第 1 版

印 次：2016 年 10 月第 1 次印刷

印 数：3000 册 定价：89.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888，88258888。

质量投诉请发邮件至 [zlts@phei.com.cn](mailto:zlts@phei.com.cn)，盗版侵权举报请发邮件至 [dbqq@phei.com.cn](mailto:dbqq@phei.com.cn)。

本书咨询联系方式：（010）51260888-819，[faq@phei.com.cn](mailto:faq@phei.com.cn)。

# 前言

有些人想学 Python，却不知如何下手；有些人已经学会 Python 的基本语法，却不知如何使用 Python 进行网站设计和开发；有些人实践过个别 Python 网络框架，却因为 Python 框架过多而无法融会贯通：本书就是为他们准备的一本指南。正所谓知识来源于实践，本书严格遵守这一原则，对每个知识点都进行了示例分析，并在第 3 篇精选了 4 个不同应用场景的网络项目，帮助读者真正掌握和运用 Python 及其周边框架。

## 为什么要读这本书

如果你不知道本书能否帮到你，或者你不知道是否要选择本书，那么请先想想在平时的学习或工作中是否遇到过这些问题：

- 有一个很好的设计网站的想法，想用 Python 实现却无从着手；
- 刚学习了编程语言的 if、for、while 等各种语法，却不知道利用编程语言到底能做些什么；
- 精通 C、C++ 等后台编程语言，却跟不上互联网蓬勃发展的新技术；
- 学了美工画图、网页设计，却不懂数据库和网站搭建；
- 觉得 Django、Tornado、Flask、Twisted 这些框架的在线资料过于晦涩难懂；
- 知道各种 Python Web 框架，却不知道自己的项目适合哪一种；
- 学过 W3CSchool 中的 Python 课程，却不知道如何使用框架提高开发效率；

- 会开发网站程序，却不知道如何集成 Nginx 等 Web 服务器；
- 听说过 SSL，让自己的网站支持 HTTPS/SSL 却力不从心；
- 学过网络编程，却还是不知道 IPv6 和 IPv4 的区别；
- 会网络数据库开发，却分不清楚 PostgreSQL、SQLite、Oracle、MySQL、SQL Server 的区别；
- 纠结于高网络流量时使用什么框架开发网站最好。

如果这其中有些是你正在困惑的地方，那么本书也许能帮到你；如果通过学习本书能帮你解决实际问题，那么笔者也就实现了写作本书的目标。

## 本书的编写特点

### 1. 零基础要求

在学习本书之前不需要具备任何计算机专业背景，任何有志于 Python 及 Web 站点设计的读者都能利用本书从头学起。本书在基础和实践部分都有大量实例，代码短小精练，紧扣所讲要点的本质，以加深读者的印象；同时结合笔者多年使用 Python 语言的开发经验，阐述了很多代码编写技巧，读者可将代码复制到自己的机器上进行实验，自行实践和演练。

### 2. 合理的章节安排

本书首先讲解了 Python 编程语言、网络和数据库基础知识、前端页面基础等知识点，然后详细讲解了 Django、Tornado、Flask、Twisted 这 4 大主流的 Python Web 开发框架，最后通过项目实践帮助读者综合运用之前学到的知识。

### 3. 最新的框架版本

主流 Python Web 框架都是开源软件，并且仍随着计算机软硬件的进步处于不断发展中，所以使用 Python 框架的开发者必须紧跟最新的框架版本！本书讲解的 4 个 Web 框架都基于最新的框架版本，使得读者能马上将其运用在当前开发环境中。这是一本内容新颖、全面的 Python Web 框架应用实战教材。

### 4. 内容全面

本书使得 Python 开发者不再局限于某个 Web 框架，一起学习这些框架有助于在学习的过程中举一反三、融会贯通。读者学完本书后可以成为 Python Web 编程方面的集大成者，对不同

网络应用场景的设计和开发都能做到得心应手。

### 5. 中小示例、项目案例，一个都不能少

根据作者多年的项目经验，本书通过将典型的示例与知识点加以整合，让读者对每章的知识点都有整体把握。最后 4 章介绍的项目案例不仅可以让读者在实际应用中更加熟练地掌握前面讲到的知识点，更能让读者了解前端开发中由轮廓到细节的完整实现流程。

本书以 Python Web 实战为主，所有代码均通过笔者上机调试，力求读者能学得懂、练得会。

## 本书的内容安排

本书共 3 篇 13 章，内容覆盖编程基础、Web 框架详解及开发实战。

### 第 1 篇（第 1~4 章）打好 Python 基础

系统学习 Python 编程语言，并且掌握进行网络开发必备的网络基础、数据库设计、HTML、CSS、JavaScript 等知识。本篇不仅适合新手学习，对有经验的开发者同样适用。

### 第 2 篇（第 5~9 章）详解主流 Python Web 框架

详细讲述了 Django、Tornado、Flask、Twisted 这 4 大主流 Python Web 框架的开发方法，在其中穿插学习 Python 虚环境、Nginx 服务器、SQLAlchemy、HTML 模板、HTML5 WebSocket 等通用组件和技术。站在框架这个巨人的肩膀上，我们不仅可以提高开发效率，还可以实现多人协同、风格统一。

### 第 3 篇（第 10~13 章）实战项目

分别应用 4 大主流框架实践开发不同类型的网站项目应用，模拟场景覆盖社交网站、聊天室、信息管理系统、物联网消息网关等各个方面，在其中还加入了 JavaScript、CSS、jQuery、Bootstrap 等前端关键技术的应用，使得读者通过深入浅出的学习和实践成为全能开发者。

笔者按照自身近 20 年的学习和开发经验编排了本书的章节顺序，所以推荐按顺序从第 1 章学习到第 13 章，尤其不能遗漏第 1 篇基础部分的内容。时间特别紧迫或者只想精通个别 Python Web 框架的读者，也可以在阅读第 1 篇后直接阅读所需框架在第 2 篇和第 3 篇中的相应部分。

本书知识点图



本书面对的读者

- Python 编程技术爱好者
- Django、Tornado、Flask、Twisted 项目参与者
- 网站设计人员
- 网站开发人员
- 网站后台开发人员
- Web 前端开发入门者
- 想由网页设计拓展为后台开发的设计者
- 由单机软件开发转向 Web 开发的技术人员
- 全栈开发人员
- 大中专院校的学生及各种 IT 培训学校的学生
- 希望自己动手设计站点原型的需求分析人员

## 编者推荐

本书的写作目的是确保读者能运用一些工具、框架、已有代码来提高开发效率和节约人力成本，确保读者能活学活用本书所讲解的内容。通过阅读本书，读者能知道如何设计一个网站、如何选择 Python Web 框架，以及如何快速使用框架进行应用开发。全书包含大量的实战案例和开发技巧，总结了使用 Python 进行 Web 开发时的优秀实践（Django、Tornado、Flask、Twisted、SQLAlchemy、Nginx、JavaScript、jQuery），讨论了各种实际问题的解决方案，是目前市场上全面实践 Python Web 开发的书籍。

## 致谢

笔者要把本书献给笔者的父母、岳父、岳母、妻子和孩子，感谢他们一直鼓励笔者，没有他们的支持，笔者无法做到这一切；还要感谢笔者的朋友和同事，感谢他们对笔者不断地鼓励和帮助。笔者非常幸运，能够和这些聪明、投入的人一起工作和交流。

# 目 录

## 第 1 篇 打好 Python 基础

第 1 章 Python 基础知识	2
1.1 Python 综述	3
1.1.1 了解 Python 的特性及版本	3
1.1.2 安装 Python	5
1.1.3 使用 Python 原生编辑器	7
1.1.4 使用 Eclipse 开发环境	9
1.1.5 Python 编程入门——解决“斐波那契数列”问题	14
1.2 数据类型	16
1.2.1 Number 类型	17
1.2.2 Sequence 类型簇	20
1.2.3 String 类型	22
1.2.4 Tuple 类型	28
1.2.5 List 类型	29
1.2.6 Set 类型	30
1.2.7 Dictionary 类型	33
1.3 流程控制	35
1.3.1 程序块与作用域	35
1.3.2 判断语句	36
1.3.3 循环语句	38
1.3.4 语句嵌套	41
1.4 函数	42
1.4.1 定义与使用	42



1.4.2 变长参数	44
1.4.3 匿名函数	46
1.5 异常	48
1.5.1 处理异常	48
1.5.2 自定义异常	50
1.6 面向对象编程	51
1.6.1 什么是面向对象	52
1.6.2 类和对象	53
1.6.3 继承	61
1.7 本章总结	64
<b>第 2 章 Web 编程之网络基础</b>	<b>65</b>
2.1 TCP/IP 网络	66
2.1.1 计算机网络综述	66
2.1.2 TCP 和 UDP	70
2.1.3 C/S 及 B/S 架构	73
2.2 HTTP	74
2.2.1 HTTP 流程	75
2.2.2 HTTP 消息结构	76
2.2.3 HTTP 请求方法	80
2.2.4 基于 HTTP 的网站开发	80
2.3 Socket 编程	82
2.3.1 Socket 基础	82
2.3.2 实战演练: Socket TCP 原语	83
2.3.3 实战演练: Socket UDP 原语	87
2.4 本章总结	88
<b>第 3 章 客户端的编程技术</b>	<b>89</b>
3.1 HTML	90
3.1.1 HTML 介绍	90
3.1.2 HTML 基本标签	93
3.1.3 HTML 表单	99
3.2 CSS	102
3.2.1 样式声明方式	103
3.2.2 CSS 语法	104
3.2.3 基于 CSS+DIV 的页面布局	106

3.3	JavaScript	108
3.3.1	在 HTML 中嵌入 JavaScript	108
3.3.2	JavaScript 的基本语法	109
3.3.3	DOM 及其读写	115
3.3.4	window 对象	118
3.3.5	HTML 事件处理	121
3.4	jQuery	124
3.4.1	使用 jQuery	124
3.4.2	选择器	125
3.4.3	行为	126
3.5	本章总结	130
第 4 章	数据库及 ORM	131
4.1	数据库概念	131
4.1.1	Web 开发中的数据库	132
4.1.2	关系数据库建模	134
4.2	关系数据库编程	137
4.2.1	常用 SQL 语句	137
4.2.2	实战演练：在 Python 中应用 SQL	143
4.3	ORM 编程	145
4.3.1	ORM 理论基础	145
4.3.2	Python ORM 库介绍	147
4.3.3	实战演练：Peewee 库编程	148
4.4	本章总结	151

第 2 篇 详解主流 Python Web 框架

第 5 章	Python 网络框架纵览	154
5.1	网络框架综述	155
5.1.1	网络框架及 MVC 架构	155
5.1.2	4 种 Python 网络框架：Django、Tornado、Flask、Twisted	156
5.2	开发环境准备	157
5.2.1	使用 Python 虚环境	157
5.2.2	Windows 环境下的安装	159
5.2.3	Linux 环境下的安装	162
5.2.4	easy_install 与 pip 的使用	163

5.3	Web 服务器	165
5.3.1	实战演练 1: WSGI 接口	165
5.3.2	实战演练 2: Linux+Nginx+uWSGI 配置	166
5.3.3	实战演练 3: 建立安全的 HTTPS 网站	172
5.4	本章总结	174
第 6 章	企业级开发框架——Django	175
6.1	Django 综述	176
6.1.1	Django 的特点及结构	176
6.1.2	安装 Django	177
6.2	实战演练: 开发 Django 站点	177
6.2.1	建立项目	177
6.2.2	建立应用	178
6.2.3	基本视图	179
6.2.4	内置 Web 服务器	181
6.2.5	模型类	182
6.2.6	表单视图	185
6.2.7	使用管理界面	188
6.3	Django 模型层	190
6.3.1	基本操作	190
6.3.2	关系操作	198
6.3.3	面向对象 ORM	202
6.4	Django 视图层	205
6.4.1	URL 映射	205
6.4.2	视图函数	211
6.4.3	模板语法	213
6.5	使用 Django 表单	218
6.5.1	表单绑定状态	219
6.5.2	表单数据验证	219
6.5.3	检查变更字段	222
6.6	个性化管理员站点	222
6.6.1	模型	222
6.6.2	模板	225
6.6.3	站点	227
6.7	本章总结	229

第 7 章 高并发处理框架——Tornado	230
7.1 Tornado 概述	231
7.1.1 Tornado 介绍	231
7.1.2 安装 Tornado	232
7.2 异步及协程基础	232
7.2.1 同步与异步 I/O	233
7.2.2 Python 关键字 yield	234
7.2.3 协程	236
7.3 实战演练：开发 Tornado 网站	239
7.3.1 网站结构	240
7.3.2 路由解析	241
7.3.3 RequestHandler	242
7.3.4 异步化及协程化	248
7.4 用户身份验证框架	250
7.4.1 安全 Cookie 机制	250
7.4.2 用户身份认证	252
7.4.3 防止跨站攻击	254
7.5 HTML5 WebSocket 概念及应用	256
7.5.1 WebSocket 概念	256
7.5.2 服务端编程	259
7.5.3 客户端编程	261
7.6 Tornado 网站部署	263
7.6.1 调试模式	264
7.6.2 静态文件	265
7.6.3 运营期配置	267
7.7 本章总结	269
第 8 章 支持快速建站的框架——Flask	270
8.1 Flask 综述	271
8.1.1 Flask 的特点	271
8.1.2 在 Windows 中的安装	272
8.1.3 在 Linux 及 Mac 中的安装	274
8.2 实战演练：开发 Flask 站点	276
8.2.1 Hello World 程序	276
8.2.2 模板渲染	278

8.2.3	重定向和错误处理	280
8.3	路由详解	281
8.3.1	带变量的路由	281
8.3.2	HTTP 方法绑定	283
8.3.3	路由地址反向生成	284
8.4	使用 Context 上下文	285
8.4.1	会话上下文	286
8.4.2	应用全局对象	287
8.4.3	请求上下文	289
8.4.4	回调接入点	291
8.5	Jinja2 模板编程	292
8.5.1	Jinja2 语法	293
8.5.2	使用过滤器	294
8.5.3	流程控制	297
8.5.4	模板继承	300
8.6	SQLAlchemy 数据库编程	303
8.6.1	SQLAlchemy 入门	303
8.6.2	主流数据库的连接方式	307
8.6.3	查询条件设置	308
8.6.4	关系操作	311
8.6.5	级联	315
8.7	WTForm 表单编程	321
8.7.1	定义表单	321
8.7.2	显示表单	322
8.7.3	获取表单数据	324
8.8	本章总结	326
第 9 章	底层自定义协议网络框架——Twisted	327
9.1	Twisted 综述	328
9.1.1	框架概况	328
9.1.2	安装 Twisted 及周边组件	328
9.2	实战演练：开发 TCP 广播系统	330
9.2.1	广播服务器	330
9.2.2	广播客户端	332
9.3	UDP 编程技术	335
9.3.1	实战演练 1：普通 UDP	335

9.3.2	实战演练 2: Connected UDP	338
9.3.3	实战演练 3: 组播技术	340
9.4	Twisted 高级话题	341
9.4.1	延迟调用	341
9.4.2	使用多线程	347
9.4.3	安全信道	349
9.5	本章总结	352

### 第 3 篇 实战项目

第 10 章	实战 1: 用 Django+PostgreSQL 开发移动 Twitter	354
10.1	项目概览	355
10.1.1	项目来源 (GitHub)	355
10.1.2	安装 PostgreSQL 数据库并配置 Python 环境	356
10.1.3	项目结构	358
10.2	页面框架设计	360
10.2.1	基模板文件	360
10.2.2	手机大小自适应 (jQuery 技术)	363
10.2.3	文本国际化	364
10.2.4	网站页面一览	367
10.3	用户注册及登录	368
10.3.1	页面设计	368
10.3.2	模型层	370
10.3.3	视图设计	371
10.4	手机消息的发布和浏览	376
10.4.1	页面设计	376
10.4.2	模型层	381
10.4.3	视图设计	382
10.5	社交朋友圈	385
10.5.1	页面设计	385
10.5.2	模型层	387
10.5.3	视图设计	388
10.6	个人资料配置	392
10.6.1	页面设计	392
10.6.2	图片上传 (第三方库 PIL)	394
10.7	Web 管理站点	397
10.7.1	定义可管理对象	397

10.7.2	配置管理员	398
10.7.3	使用管理站点	398
10.8	本章总结	400
<b>第 11 章 实战 2: 用 Tornado+jQuery 开发 WebSocket 聊天室</b>		401
11.1	聊天室概览	402
11.1.1	项目介绍	402
11.1.2	安装和代码结构	403
11.2	消息通信	404
11.2.1	建立网站	404
11.2.2	WebSocket 服务器	407
11.2.3	WebSocket 客户端	408
11.3	聊天功能	411
11.3.1	昵称	411
11.3.2	消息来源	413
11.3.3	历史消息缓存	415
11.4	用户面板	416
11.4.1	用 CSS 定义用户列表	416
11.4.2	服务器通知	418
11.4.3	响应服务器动态通知 (jQuery 动态编程)	419
11.5	本章总结	420
<b>第 12 章 实战 3: 用 Flask+Bootstrap+Restful 开发学校管理系统</b>		421
12.1	系统概览	422
12.1.1	来源及功能	422
12.1.2	项目安装	423
12.1.3	代码结构	425
12.2	数据模型设计	427
12.2.1	E-R 图设计	427
12.2.2	SQLAlchemy 建模	429
12.3	响应式页面框架设计	435
12.3.1	基模板组件引用	435
12.3.2	响应式导航	439
12.4	新建学校	441
12.4.1	WTForm 表单	441
12.4.2	视图及文件上传	443

12.4.3	响应式布局	445
12.5	学校管理	447
12.5.1	查询视图	447
12.5.2	分页模板	450
12.6	Restful 接口	453
12.6.1	Restful 概念	453
12.6.2	Restless 插件	454
12.6.3	开发 Restful 接口	457
12.7	本章总结	461
第 13 章	实战 4：用 Twisted+SQLAlchemy+ZeroMQ 开发跨平台物联网消息网关	463
13.1	项目概况	464
13.1.1	功能定义	464
13.1.2	安装和测试	465
13.1.3	项目结构	468
13.2	项目设计	469
13.2.1	SQLAlchemy 建模	469
13.2.2	TCP 接口设计	474
13.3	通信引擎	476
13.3.1	跨平台安全端口	477
13.3.2	管理连接	478
13.3.3	收发数据	479
13.3.4	TCP 流式分包	482
13.3.5	异步执行	484
13.4	协议编程	486
13.4.1	执行命令	486
13.4.2	struct 解析字节流	489
13.4.3	序列号生成	490
13.4.4	连接保持	491
13.4.5	发送 Response	492
13.4.6	错误机制	494
13.5	ZeroMQ 集群	496
13.5.1	内部接口设计	496
13.5.2	PUB/SUB 通信模型编程	498
13.6	本章总结	499



# 5

## 第 5 章

---

### Python 网络框架纵览

从本章开始，我们进入使用 Python 进行 Web 应用程序开发的框架学习阶段。目前 Python 的网络编程框架已经多达几十个，逐个学习它们显然不现实。但这些框架在系统架构和运行环境中有很多共通之处，本章带领读者学习所有基于 Python 网络框架开发的常用知识，了解这些内容对之后学习具体的框架，能够起到事半功倍的作用。

本章主要涉及的知识点如下。

- **Python 网络框架综述：**了解什么是网络框架，分析 Python 最主要的网络框架的特点及适用环境，学习 Web 开发中经典的 MVC 架构。
- **组件安装准备：**学习 Python 虚环境的概念和作用，以及 Python 在 Windows 和 Linux 中安装虚环境和第三方组件的通用方法。
- **网络开发通用工具：**Python 网络开发标准接口 WSGI、网络客户端调试工具等。
- **Web 服务器：**Nginx 的安装及配置及安全的 HTTPS 站点的搭建方法。

## 5.1 网络框架综述

本节学习 Python 网络框架的概念和开发方法,然后了解 Web 经典架构 MVC 的原理和作用。

### 5.1.1 网络框架及 MVC 架构

所谓网络框架是指这样的一组 Python 包,它能够使开发者专注于网站应用业务逻辑的开发,而无须处理网络应用底层的协议、线程、进程等方面。这样能大大提高开发者的工作效率,同时提高网络应用程序的质量。

在目前 Python 语言的几十个开发框架中,几乎所有的全栈网络框架都强制或引导开发者使用 MVC 架构开发 Web 应用。所谓全栈网络框架,是指除了封装网络和线程操作,还提供 HTTP 栈、数据库读写管理、HTML 模板引擎等一系列功能的网络框架。本书重点讲解的 Django、Tornado 和 Flask 是全栈网络框架的典型标杆;而 Twisted 更专注于网络底层的高性能封装而不提供 HTML 模板引擎等界面功能,所以不能称之为全栈框架。

MVC (Model-View-Controller) 模式最早由 Trygve Reenskaug 在 1978 年提出,在 20 世纪 80 年代是程序语言 Smalltalk 的一种内部架构。后来 MVC 被其他语言所借鉴,成为了软件工程中的一种软件架构模式。MVC 把 Web 应用系统分为 3 个基本部分。

- 模型 (Model): 用于封装与应用程序的业务逻辑相关的数据及对数据的处理方法,是 Web 应用程序中用于处理应用程序的数据逻辑的部分,Model 只提供功能性的接口,通过这些接口可以获取 Model 的所有功能。Model 不依赖于 View 和 Controller,它们可以在任何时候调用 Model 访问数据。有些 Model 还提供了事件通知机制,为在其上注册过的 View 或 Controller 提供实时的数据更新。
- 视图 (View): 负责数据的显示和呈现,View 是对用户的直接输出。MVC 中的一个 Model 通常为多个 View 提供服务。为了获取 Model 的实时更新数据,View 应该尽早地注册到 Model 中。
- 控制器 (Controller): 负责从用户端收集用户的输入,可以看成提供 View 的反向功能。当用户的输入导致 View 发生变化时,这种变化必须是通过 Model 反映给 View 的。在 MVC 架构下,Controller 一般不能与 View 直接通信,这样提高了业务数据的一致性,即以 Model 作为数据中心。

这 3 个基本部分互相分离，使得在改进和升级界面及用户交互流程时，不需要重写业务逻辑及数据访问代码。MVC 架构如图 5.1 所示。

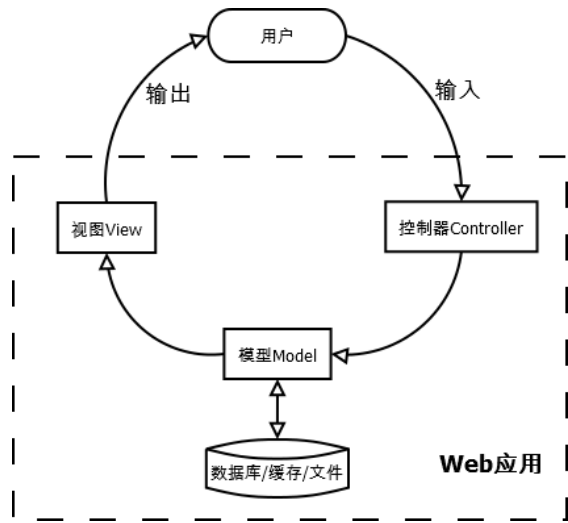


图 5.1 MVC 架构图

注意：MVC 在除 Python 外的其他语言中也有广泛应用，例如 VC++ 的 MFC、Java 的 Struts 及 Spring、C# 的 .NET 开发框架，读者应该有深刻的体会。

### 5.1.2 4 种 Python 网络框架：Django、Tornado、Flask、Twisted

Python 作为最主要的互联网语言，在其发展的二十多年中出现了数十种网络框架，例如 Django、Flask、Twisted、Bottle、Web.py 等，它们有的历史悠久，有的蓬勃发展，而有的已被停止维护，如何对其进行取舍常常使得初学者犹豫不决。本书带领读者学习当今主流的 4 种 Python 网络框架。

#### 1 . Django

Django 发布于 2003 年，是当前 Python 世界上最负盛名且最成熟的网络框架。最初用来制作在线新闻的 Web 站点，目前已发展为应用最广泛的 Python 网络框架。

Django 的各模块之间结合得比较紧密，所以在功能强大的同时又是一个相对封闭的系统，但是其健全的在线文档及开发社区，使开发者在遇到问题时能找到解决方法。

## 2 . Tornado

Tornado 是一个强大的、支持协程、高效并发且可扩展的 Web 服务器，发布于 2009 年 9 月，应用于 FriendFeed、Facebook 等社交网站。Tornado 的强项在于可以利用它的异步协程机制开发高并发的服务器系统。

## 3 . Flask

Flask 是 Python Web 框架族里比较年轻的一个，发布于 2010 年，这使得它吸收了其他框架的优点并且把自己的主要领域定义在了微小项目上。

## 4 . Twisted

Twisted 是一个有着十多年历史的开源事件驱动框架。Twisted 不像前 3 种着眼于网络 HTTP 应用开发，而是适用于从传输层到自定义应用协议的所有类型的网络程序的开发，并能不同的操作系统上提供很高的运行效率。

说明：目前 Twisted 对 Python 3 的支持比较有限，所以用 Twisted 开发的应用建议使用 Python 2.7 版本。

## 5.2 开发环境准备

在 Python 中进行网络框架开发的第 1 步是安装所使用的组件。Python 有两种安装组件的方法，分别是 easy\_install 安装和 pip 安装。

easy\_install 出现较早，而 pip 是 easy\_install 的改进版，提供了更好的提示信息。在一般情况下，比较老（2000 年之前）的 Python 库需要用 easy\_install 进行安装，比较新的 Python 库适合用 pip 进行安装。在使用 Python 做大项目开发之前，开发者应该同时安装 easy\_install 和 pip。

## 5.2.1 使用 Python 虚环境

使用 Python 进行多个项目的开发时，每个项目可能会需要安装不同的组件。把这些组件安装在同一台计算机下可能会导致相互之间有冲突，比如项目 A 使用 SQLAlchemy 的 0.5 版本，而项目 B 使用 SQLAlchemy 的 0.9 版本，那么同时安装这两个版本会在使用时产生冲突。使用 Python 虚环境则可以避免这样的问题。

Python 虚环境是一套由 Ian Bicking 编写的管理独立 Python 运行环境的系统。这样，开发者或系统管理者可以让每个项目运行在独立的虚环境中，从而避免了不同项目之间组件配置的冲突。

在使用 Python 虚环境之前，需要先用 pip 安装虚环境工具包，然后就可以通过一系列虚环境命令部署和管理 Python 项目。

### 1. 虚环境的安装

因为 pip 命令在不同的操作系统中的使用方式一致，所以下面以 Linux 系统为例演示虚环境的安装及使用。可以通过如下命令一步完成对虚环境的安装：

```
#pip install virtualenv
```

### 2. 虚环境的使用

通过如下命令为一个已有的项目建立虚环境：

```
#cd [项目所在目录]
#virtualenv venv
```

该条命令执行后，将在当前目录中建立一个 venv 目录，该目录复制了一份完整的当前系统的 Python 环境。之后运行 Python 时可以直接运行该目录的 bin 文件夹中的命令。bin 文件夹的内容如图 5.2 所示。

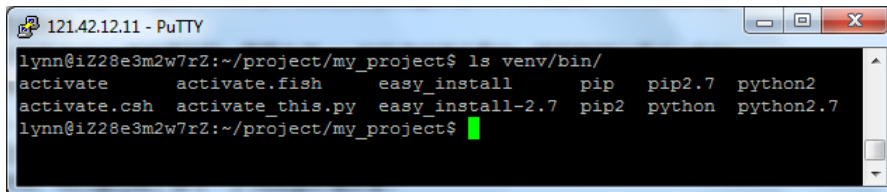


图 5.2 bin 文件夹的内容

比如，在当前虚环境下安装 Tornado 组件：

```
#./venv/bin/pip install tornado
```

则该组件将被安装在 `venv/lib` 目录中，而不会影响系统的 Python 环境。

再比如，用该虚环境运行 Python 程序：

```
#./venv/bin/python xxxx.py
```

也可以用 `activate` 命令启动虚环境，之后不必再显式地调用虚环境 `bin` 文件夹中的命令，如下命令与之前的 `/venv/bin/python` 命令的效果相同：

```
#source ./venv/bin/activate
(venv)#python xxxx.py
```

用 `deactivate` 命令可以退出用 `activate` 进入的虚环境，比如：

```
#./venv/bin/activate
(venv)# /*此处执行的命令在虚环境中运行*/
#deactivate
# /*此处已退出虚环境*/
```

注意：为保证项目之间的独立性，笔者建议所有使用 `easy_install` 和 `pip` 的组件安装都

在项目虚环境中进行。本书后面不再说明。

## 5.2.2 Windows 环境下的安装

`easy_install` 和 `pip` 不是 Python 安装包中的内容，所以需要开发者手动从网上下载适用的最新版本，然后安装在本地。

### 1. 下载 `easy_install` 和 `pip`

`easy_install` 是一个 Python 的安装包管理系统。`easy_install` 是由 PEAK (Python Enterprise Application Kit) 开发的安装 Python 模块的命令工具。而 `pip` 的功能更丰富，为安装过程提供了更多的信息。它们都可以从 Python 官方网站下载。

- `easy_install`：从 [https://pypi.python.org/pypi/ez\\_setup](https://pypi.python.org/pypi/ez_setup) 下载最新的 `easy_installer` 安装脚本

包 ez\_setup-0.9.tar.gz。

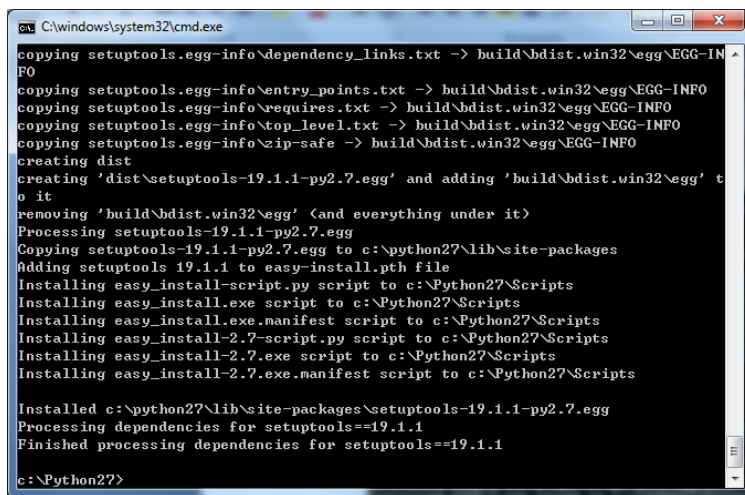
- pip: 从 <https://pypi.python.org/pypi/pip/downloads> 下载 pip 的最新版本, 例如 pip-8.0.2.tar.gz (md5, pgp)。

## 2. 安装 easy\_install

用 7zip 等压缩软件解压 tar.gz 文件后, 打开 Windows 命令提示符, 通过运行如下命令完成对 easy\_installer 的安装:

```
C:\Python27\python ez_setup.py
```

如果出现如图 5.3 所示的界面, 则安装成功。



```
C:\windows\system32\cmd.exe
copying setuptools.egg-info\dependency_links.txt -> build\bdist.win32\egg\EGG-INFO
copying setuptools.egg-info\entry_points.txt -> build\bdist.win32\egg\EGG-INFO
copying setuptools.egg-info\requires.txt -> build\bdist.win32\egg\EGG-INFO
copying setuptools.egg-info\top_level.txt -> build\bdist.win32\egg\EGG-INFO
copying setuptools.egg-info\zip-safe -> build\bdist.win32\egg\EGG-INFO
creating dist
creating 'dist\setuptools-19.1.1-py2.7.egg' and adding 'build\bdist.win32\egg' to it
removing 'build\bdist.win32\egg' (and everything under it)
Processing setuptools-19.1.1-py2.7.egg
Copying setuptools-19.1.1-py2.7.egg to c:\python27\lib\site-packages
Adding setuptools 19.1.1 to easy-install.pth file
Installing easy_install-script.py script to c:\Python27\Scripts
Installing easy_install.exe script to c:\Python27\Scripts
Installing easy_install.exe.manifest script to c:\Python27\Scripts
Installing easy_install-2.7-script.py script to c:\Python27\Scripts
Installing easy_install-2.7.exe script to c:\Python27\Scripts
Installing easy_install-2.7.exe.manifest script to c:\Python27\Scripts

Installed c:\python27\lib\site-packages\setuptools-19.1.1-py2.7.egg
Processing dependencies for setuptools==19.1.1
Finished processing dependencies for setuptools==19.1.1

c:\Python27>
```

图 5.3 easy\_installer 安装成功后的界面

## 3. 安装 pip

将下载的 pip-x.x.x.tar.gz 文件用 7zip 等压缩软件解压到某个目录中。解压后的内容如图 5.4 所示。

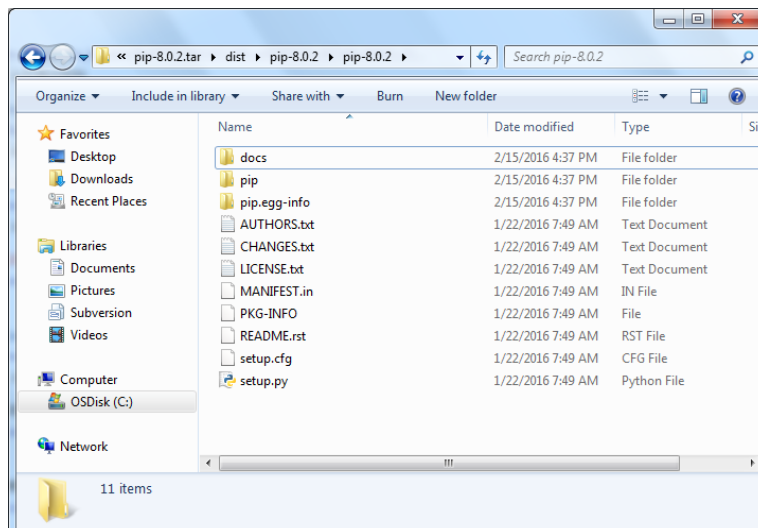


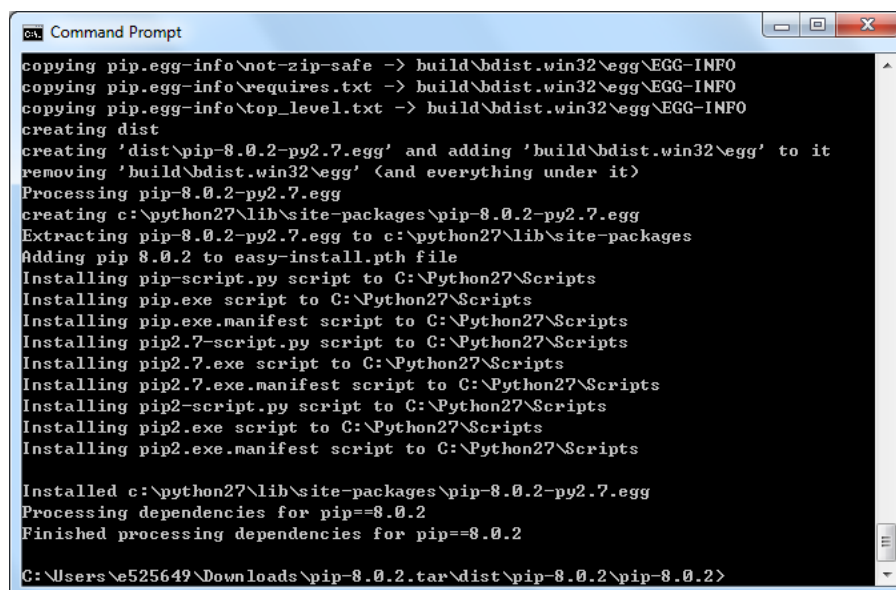
图 5.4 解压 pip 安装包

在 cmd 中进入到该目录，运行如下命令进行安装：

```
C:\...\pip-8.0.2.tar> python setup.py install
```

安装过程约为 1 分钟，出现如图 5.5 所示界面时安装完成。





```
copying pip.egg-info\not-zip-safe -> build\bdist.win32\egg\EGG-INFO
copying pip.egg-info\requires.txt -> build\bdist.win32\egg\EGG-INFO
copying pip.egg-info\top_level.txt -> build\bdist.win32\egg\EGG-INFO
creating dist
creating 'dist\pip-8.0.2-py2.7.egg' and adding 'build\bdist.win32\egg' to it
removing 'build\bdist.win32\egg' (and everything under it)
Processing pip-8.0.2-py2.7.egg
creating c:\python27\lib\site-packages\pip-8.0.2-py2.7.egg
Extracting pip-8.0.2-py2.7.egg to c:\python27\lib\site-packages
Adding pip 8.0.2 to easy-install.pth file
Installing pip-script.py script to C:\Python27\Scripts
Installing pip.exe script to C:\Python27\Scripts
Installing pip.exe.manifest script to C:\Python27\Scripts
Installing pip2.7-script.py script to C:\Python27\Scripts
Installing pip2.7.exe script to C:\Python27\Scripts
Installing pip2.7.exe.manifest script to C:\Python27\Scripts
Installing pip2-script.py script to C:\Python27\Scripts
Installing pip2.exe script to C:\Python27\Scripts
Installing pip2.exe.manifest script to C:\Python27\Scripts

Installed c:\python27\lib\site-packages\pip-8.0.2-py2.7.egg
Processing dependencies for pip==8.0.2
Finished processing dependencies for pip==8.0.2

C:\Users\525649\Downloads\pip-8.0.2.tar\dist\pip-8.0.2\pip-8.0.2>
```

图 5.5 pip 安装完成

#### 4. 配置环境变量

Easy\_install 和 pip 会安装在系统 Python 安装目录中的 scripts 子目录, 所以如果以后要顺利使用 easy\_install 命令, 则需要将其目录添加到系统的 Path 目录中, 如图 5.6 所示。

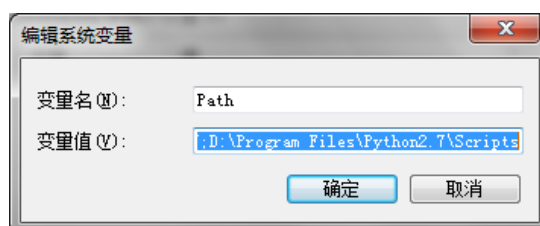


图 5.6 easy\_install 安装

技巧: 在 Windows 系统中, 路径名无须区分大小写。

### 5.2.3 Linux 环境下的安装

在 Linux 中可以直接通过命令下载并安装。

#### 1. easy\_install 的下载与安装

假设当前 easy\_install 的版本为 0.9，则通过如下命令下载和安装 easy\_install：

```
#wget https://pypi.python.org/packages/source/e/ez_setup/ez_setup-0.9.tar.gz
#tar -zxvf ez_setup-0.9.tar.gz
#cd ez_setup-0.9
#python ez_setup.py
```

注意：如果运行 python ez\_setup.py 时出现 Permission Denied 的问题，则需要用管理员的权限运行，即 sudo python ez\_setup.py。

安装成功后，会出现如图 5.7 所示的界面。

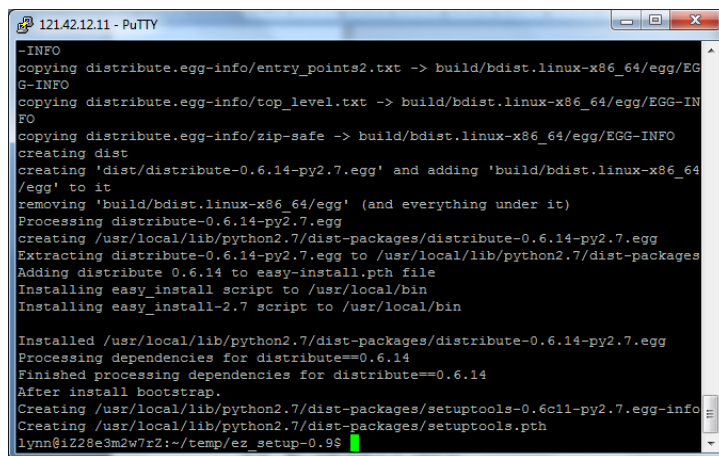


图 5.7 在 Linux 下安装 easy\_install 成功

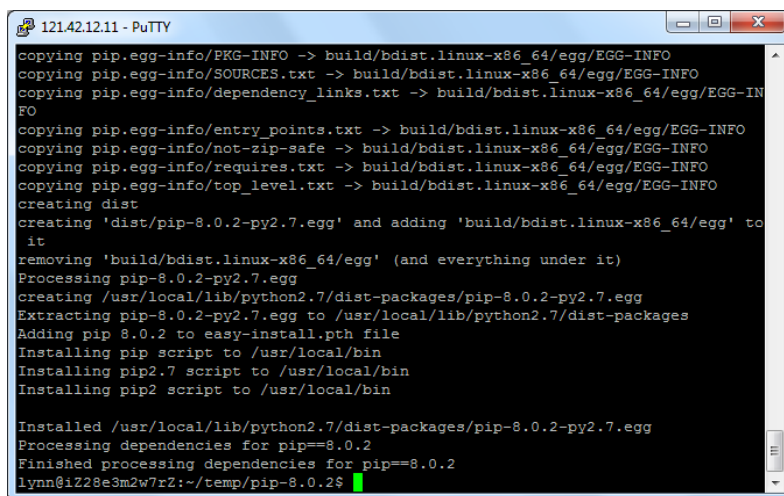
#### 2. pip 的下载与安装

假设当前 pip 的版本为 8.0.2，则可通过如下命令下载和安装 pip：

```
#wget https://pypi.python.org/packages/source/p/pip/pip-8.0.2.tar.gz
```

```
#tar -zxvf pip-8.0.2.tar.gz
#cd pip-8.0.2
#python setup.py install
```

安装成功后，出现如图 5.8 所示的界面。



```
copying pip.egg-info/PKG-INFO -> build/bdist.linux-x86_64/egg/EGG-INFO
copying pip.egg-info/SOURCES.txt -> build/bdist.linux-x86_64/egg/EGG-INFO
copying pip.egg-info/dependency_links.txt -> build/bdist.linux-x86_64/egg/EGG-INFO
copying pip.egg-info/entry_points.txt -> build/bdist.linux-x86_64/egg/EGG-INFO
copying pip.egg-info/not-zip-safe -> build/bdist.linux-x86_64/egg/EGG-INFO
copying pip.egg-info/requires.txt -> build/bdist.linux-x86_64/egg/EGG-INFO
copying pip.egg-info/top_level.txt -> build/bdist.linux-x86_64/egg/EGG-INFO
creating dist
creating 'dist/pip-8.0.2-py2.7.egg' and adding 'build/bdist.linux-x86_64/egg' to
it
removing 'build/bdist.linux-x86_64/egg' (and everything under it)
Processing pip-8.0.2-py2.7.egg
creating /usr/local/lib/python2.7/dist-packages/pip-8.0.2-py2.7.egg
Extracting pip-8.0.2-py2.7.egg to /usr/local/lib/python2.7/dist-packages
Adding pip 8.0.2 to easy-install.pth file
Installing pip script to /usr/local/bin
Installing pip2.7 script to /usr/local/bin
Installing pip2 script to /usr/local/bin

Installed /usr/local/lib/python2.7/dist-packages/pip-8.0.2-py2.7.egg
Processing dependencies for pip==8.0.2
Finished processing dependencies for pip==8.0.2
lynn@i228e3m2w7zZ:~/temp/pip-8.0.2$
```

图 5.8 在 Linux 下安装 pip 成功

## 5.2.4 easy\_install 与 pip 的使用

easy\_install 和 pip 的使用具有跨平台性，本节以 Linux 为例讲解 easy\_install 和 pip 的使用方法。

### 1. 用 easy\_install 管理其他组件

可以直接通过 easy\_installer 安装所需要的 Python 组件。本书涉及的 Django、Flask、Twisted、SQLAlchemy 等需要通过 easy\_install 进行安装和管理。下面以 Flask 组件的安装和管理为例演示 easy\_install 的使用方法。

从 Pypi 网站自动下载并安装组件：

```
#easy_install flask
```

自动升级组件：

```
#easy_install upgrade flask
```

用本地已有的 egg 文件安装组件：

```
#easy_install /my_downloads/flask-0.9.1-py2.7.egg
```

通过 -m 参数卸载组件：

```
#easy_install -m flask
```

注意：安装时需用系统管理员的权限运行。

## 2. 用 pip 管理其他组件

需要使用 pip 时，可以直接运行 pip 命令进行软件的安装和卸载。pip 比 easy\_install 的命令更丰富。以 Tornado 框架组件的安装和管理为例演示 pip 的使用方法如下。

从 Pypi 网站自动下载并安装组件：

```
#pip install tornado
#pip install tornado= 1.0.8 //可以在安装时指定版本
```

自动升级组件：

```
#pip install -U tornado
```

升级组件到指定的版本：

```
#pip install -U tornado=1.0.9
```

找到 Pypi 网站中所有与某关键字有关系的组件：

```
#pip search framework //查看所有与"framework"关键字相关的组件
```

卸载组件：

```
#pip uninstall tornado
```

查看所有选项：

```
#pip help
```

在 Windows 中，pip 的调用方式与 Linux 略有不同，需要通过 python -m pip 的方式调用。

比如下面是 Windows 中用 pip 安装 Tornado 的命令：

```
C:\Python2\Scripts\>python -m pip install tornado
```

## 5.3 Web 服务器

Web 服务器是连接用户浏览器与 Python 服务器端程序的中间节点，在网站建立的过程中起着重要的作用。目前最主流的 Web 服务器包括 Nginx、Apache、lighttpd、IIS 等。Python 服务器端程序在 Linux 平台下使用最广泛的是 Nginx。本节学习 Python 程序与 Web 服务器连接的 WSGI 接口、Nginx 的安装和配置方法，以及搭建 SSL 网站的技术。

### 5.3.1 实战演练 1：WSGI 接口

WSGI 是将 Python 服务器端程序连接到 Web 服务器的通用协议。由于 WSGI 的通用性，出现了独立的 WSGI 程序，例如 uWSGI 和 Apache 的 mod\_wsgi。

WSGI 的全称为 Web Server Gateway Interface，也可称作 Python Web Server Gateway Interface，开始于 2003 年，为 Python 语言定义 Web 服务器和服务器端程序的通用接口规范。因为 WSGI 在 Python 中的成功，所以其他语言诸如 Perl 和 Ruby 也定义了类似 WSGI 作用的接口规范。WSGI 的作用如图 5.9 所示。

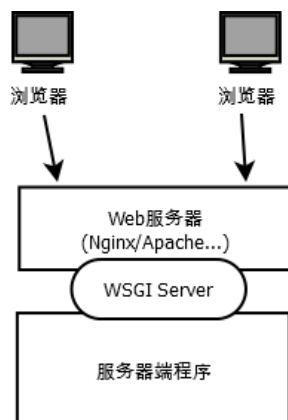


图 5.9 WSGI 的作用

从图 5.9 中可见 WSGI 的接口分为两个：一个是与 Web 服务器的接口，另一个是与服务器

端程序的接口。WSGI Server 与 Web 服务器的接口包括 uwsgi、fast cgi 等，服务器端程序的开发者无须学习这部分的详细内容。服务器端的开发者需要关注的是 WSGI 与服务器程序的接口。

WSGI 的服务器程序的接口非常简单，以下是一个服务器端程序的例子，将该文件保存为 webapp.py:

```
def application(environ, start_response):
    start_response('200 OK', [('Content-Type', 'text/html')])
    return '<b>Hello, world!</b>'
```

该代码只定义了一个函数 app，所有来自 Web 服务器的 HTTP 请求都会由 WSGI 服务转换为对该函数的调用。该示例的 app 函数中没有复杂的处理，只是通过 start\_response 返回了状态码，并通过 return 返回了一个固定的 HTTP 消息体。与该服务器端程序相对应的是下面的 WSGI Server 程序:

```
#引入Python的WSGI包
from wsgiref.simple_server import make_server
#引入服务器端程序的代码
from webapp import application

#实例化一个监听 8080 端口的服务器
server = make_server('', 8080, application)
# 开始监听 HTTP 请求:
server.serve_forever()
```

将该 WSGI Server 的程序保存为 wsgi\_server.py，通过下面的命令即可启动一个 Web 服务器，该服务器对所有的请求都返回 Hello World 页面:

```
#python wsgi_server.py
```

注意：虽然 WSGI 的设计目标是连接标准的 Web 服务器 ( Nginx、Apache 等 ) 与服务

器端程序，但 WSGI Server 本身也可以作为 Web 服务器运行。由于性能方面的

原因，该服务器一般只做测试使用，不能用于正式运行。

### 5.3.2 实战演练 2: Linux+Nginx+uWSGI 配置

Nginx 是由俄罗斯工程师开发的一个高性能 HTTP 和反向代理服务器，其第 1 个公开版本 0.1.0 于 2004 年以开源形式发布。自发布后，它以运行稳定、配置简单、资源消耗低而闻名。许多知名网站（百度、新浪、腾讯等）均采用 Nginx 作为 Web 服务器。

因为 Nginx 是 Python 在 Linux 环境下的首选 Web 服务器之一，所以本节以 Ubuntu Linux 为例演示 Nginx 的安装及配置方法。

#### 1. 安装 Nginx

在 Ubuntu Linux 中可以通过如下命令安装 Nginx：

```
#sudo apt-get install nginx
```

安装结果如图 5.10 所示。

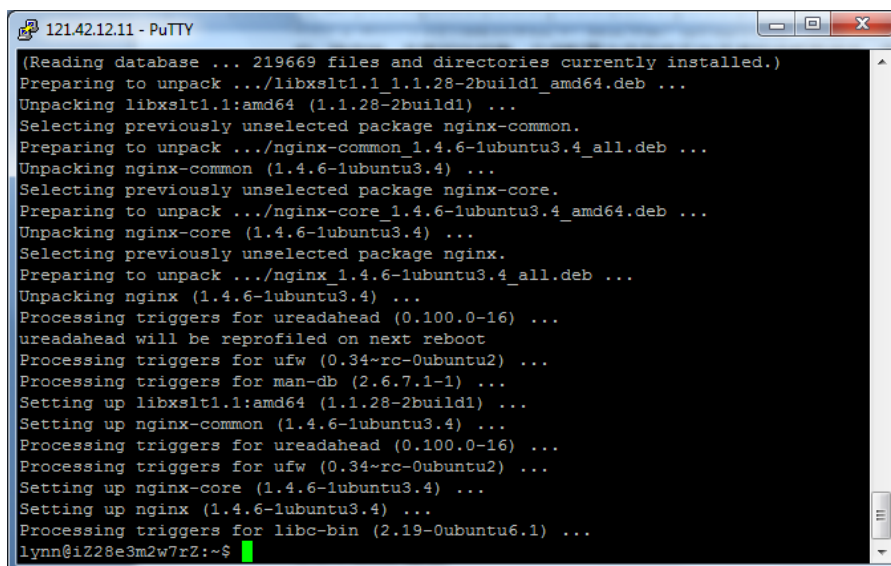


图 5.10 Linux 安装成功

安装程序把 Nginx 以服务的形式安装在系统中，相关的程序及文件路径如下。

- 程序文件：放在/usr/sbin/nginx 目录中。



- 全局配置文件：/etc/nginx/nginx.conf。
- 访问日志文件：/var/log/nginx/access.log。
- 错误日志文件：/var/log/nginx/error.log。
- 站点配置文件：/etc/nginx/sites-enabled/default。

安装好后，可以通过如下命令启动 Nginx 服务器：

```
sudo service nginx start
```

停止 Nginx 服务器：

```
sudo service nginx stop
```

查看 Nginx 服务的状态：

```
sudo service nginx status
```

重启 Nginx 服务器：

```
sudo service nginx restart
```

## 2 . Nginx 配置文件

Nginx 安装后以默认方式启动，在开发调试的过程中可能需要调整 Nginx 的运行参数，这些运行参数通过全局配置文件（nginx.conf）和站点配置文件（sites-enabled/\*）进行设置。对全局配置文件（/etc/nginx/nginx.conf）中的关键可设置参数解析如下：

```
user www-data;                                ##定义运行 Nginx 的用户

worker_processes 4;                            ##Nginx 进程数，应设置与系统 CPU 数量相等的数值

worker_rlimit_nofile 65535;                   ##每个 Nginx 进程可以打开的最大文件数

events {
    worker_connections 768;                    ##每个 Nginx 进程允许的最大客户端连接数

    #在 Nginx 接到一个新连接通知后调用 accept() 来接受尽量多的连接
    multi_accept off;
}

http {
```

```

##
# Basic Settings
##

sendfile on;                ##是否允许文件上传
client_header_buffer_size 32k;  ##上传文件大小限制
tcp_nopush on;              ##防止网络阻塞
tcp_nodelay on;             ##防止网络阻塞
keepalive_timeout 65;        ##允许的客户端长连接最大秒数

##Nginx 散列表大小。本值越大，占用的内存空间越大，但路由速度越快
types_hash_max_size 2048;

access_log /var/log/nginx/access.log;    ##访问日志文件路径名
error_log /var/log/nginx/error.log;      ##错误日志文件路径名

## 如下两条用 include 命令加载站点配置文件
include /etc/nginx/conf.d/*.conf;
include /etc/nginx/sites-enabled/*;
}

```

在每个 Nginx 服务器中可以运行多个 Web 站点，每个站点的配置通过站点配置文件设置。每个站点应该以一个单独的配置文件存放在/etc/nginx/sites-enabled 目录中，默认站点的配置文件名为/etc/nginx/sites-enabled/default，对其中关键内容的解析如下：

```

server {

    ##配置站点监听的端口
    listen 80;

    root /usr/share/nginx/html;          ##配置 HTTP 根页面目录
    index index.html index.htm;          ##配置 HTTP 根目录中的默认页面

    #站点监听的 IP 地址，默认的 localhost 只可用于本机访问，一般需要将其更改为真实 IP
    server_name localhost;

    ##location 用于配置 URL 的转发接口
    location /user/ {
        ##此处配置 http://server_name/user/ 的转发地址
        proxy_pass http://127.0.0.1:8080;
    }
}

```

```
##错误页面配置，如下配置定义 HTTP 404 错误的显示页面为/404.html
error_page 404 /404.html;
}
```

### 3. 安装 uWSGI 及配置

uWSGI 是 WSGI 在 Linux 中的一种实现，这样开发者就无须自己编写 WSGI Server 了。

使用 pip 命令可以直接安装 uWSGI:

```
#pip install uwsgi
```

安装完成后即可运行 uwsgi 命令启动 WSGI 服务器，uwsgi 命令通过启动参数的方式配置可选的运行方式。比如，如下命令可以运行 uWSGI，用于加载之前编写的服务器端程序 webapp.py:

```
#uwsgi --http:9090 --wsgi-file webapp.py
*** Starting uWSGI 2.0.12 (64bit) on [Wed Feb 17 15:27:21 2016] ***
compiled with version: 4.8.2 on 17 February 2016 15:21:40
os: Linux-3.13.0-32-generic #57-Ubuntu SMP Tue Jul 15 03:51:08 UTC 2014
nodename: iz28e3m2w7rZ
machine: x86_64
clock source: unix
pcre jit disabled
detected number of CPU cores: 1
current working directory: /home/lynn/project/my_project
detected binary path: /home/lynn/project/my_project/venv/bin/uwsgi
*** WARNING: you are running uWSGI without its master process manager ***
your processes number limit is 3746
your memory page size is 4096 bytes
detected max file descriptor number: 65535
lock engine: pthread robust mutexes
thunder lock: disabled (you can enable it with --thunder-lock)
uWSGI http bound on 9090 fd 4
spawned uWSGI http 1 (pid: 18669)
uwsgi socket 0 bound to TCP address 127.0.0.1:34755 (port auto-assigned) fd 3
Python version: 2.7.6 (default, Jun 22 2015, 18:01:27) [GCC 4.8.2]
*** Python threads support is disabled. You can enable it with --enable-threads ***
Python main interpreter initialized at 0x1ff3b20
your server socket listen backlog is limited to 100 connections
your mercy for graceful operations on workers is 60 seconds
mapped 72768 bytes (71 KB) for 1 cores
*** Operational MODE: single process ***
```

```
WSGI app 0 (mountpoint='') ready in 0 seconds on interpreter 0x1ff3b20 pid: 18668 (default
app)
*** uWSGI is running in multiple interpreter mode ***
spawned uWSGI worker 1 (and the only) (pid: 18668, cores: 1)
```

启动时用 `--http` 参数指定了监听端口，用 `--wsgi-file` 指定了服务器端的程序名。如上所示，uWSGI 在启动的过程中会输出系统的一些环境信息：服务器名、进程数限制、服务器硬件配置、最大文件句柄数等。

除了在 uWSGI 启动命令行中提供配置参数，uWSGI 还允许通过一个配置文件设置这些配置参数，比如可以编写如下配置文件，保存在文件名 `uwsgi.ini` 中：

```
[uwsgi]
http = 9090
wsgi-file = webapp.py
```

启动 uWSGI 时直接指定配置文件即可：

```
#uwsgi uwsgi.ini
```

此时用浏览器访问服务器的 9090 端口，效果如图 5.11 所示。

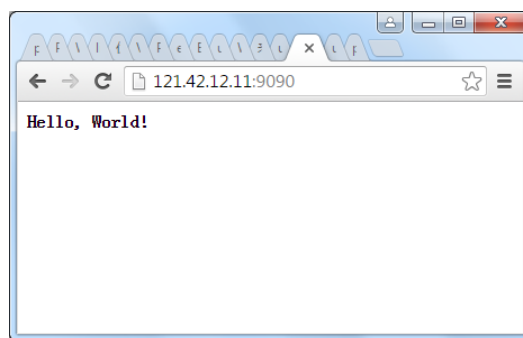


图 5.11 uWSGI 服务运行的效果

除了 `http` 和 `wsgi-file` 参数，uWSGI 还有很多其他参数，常用的如下。

- **socket**: 以 WSGI 的 Socket 方式运行，并指定连接地址和端口。该 Socket 接口是 uWSGI 与其他 Web 服务器（Nginx/Apache）等进行对接的方式。
- **chdir**: 指定 uWSGI 启动后的当前目录。
- **processes**: 指定启动服务器端程序的进程数。

- **threads:** 指定每个服务器端程序的线程数。即服务器端的总线程数为 `processes×threads`。
- **uid:** 指定运行 uWSGI 的 Linux 用户 id。

比如，如下配置文件用于用 Socket 方式启动一个 uWSGI 服务器，并配置了进程和线程数：

```
uwsgi]
socket = 127.0.0.1: 3011
wsgi-file = webapp.py
processes = 4
threads = 3
```

#### 4 . 集成 Nginx 与 uWSGI

直接通过在站点配置文件中为 location 配置 `uwsgi_pass`，即可将 Nginx 与 uWSGI 集成，建立一个基于 Nginx+Python 的正式站点。针对如下 uWSGI 接口有：

```
uwsgi]
socket = 127.0.0.1: 3011
wsgi-file = webapp.py
```

Nginx 的站点配置文件为：

```
server {
    listen 80;

##此处改为服务器的真实 IP
    server_name 121.12.134.11;

    location /{
        ##此处 IP 与 Port 配置必须与 uwsgi 接口中参数相同
        uwsgi_pass http://127.0.0.1:3011;
    }
}
```

技巧：可以为一个 uWSGI 配置多个 Nginx Server 和 location，这样就轻松实现了以多

域名访问同一个 Python 程序。

### 5.3.3 实战演练 3：建立安全的 HTTPS 网站

普通 HTTP 站点的协议与数据以明文方式在网络上传输，而 HTTPS（Hypertext Transfer Protocol over Secure Socket Layer）是以安全为目标的 HTTP 通道，即在 HTTP 下加入 SSL 层，通过 SSL 达到数据加密及身份认证的功能。目前几乎所有的银行、证券、公共交通的网站均以 HTTPS 方式搭建。

OpenSSL 是一个强大的免费 Socket 层密码库，蕴含了主要的密码算法、常用的密钥和证书封装管理功能及 SSL 协议。目前大多数网站通过 OpenSSL 工具包搭建 HTTPS 站点，其步骤如下。

- 在服务器中安装 OpenSSL 工具包。
- 生成 SSL 密钥和证书。
- 将证书配置到 Web 服务器。
- 在客户端安装 CA 证书。

本节演示在 Linux Ubuntu 下 OpenSSL 的使用方法，以及 Nginx 在 Linux 下的证书配置方式。Windows 中 OpenSSL 的使用方式与 Linux 中的完全一致，读者可以自行尝试。

#### 1. 在服务器中安装 OpenSSL 工具包

通过如下两条命令安装 OpenSSL：

```
#sudo apt-get install openssl
#sudo apt-get install libssl-dev
```

命令运行成功后，OpenSSL 命令和配置文件将被安装到 Linux 系统目录中。

- OpenSSL 命令：/usr/bin/openssl。
- 配置文件：/usr/lib/ssl/\*。

#### 2. 生成 SSL 密钥和证书

通过如下步骤生成 CA 证书 ca.crt、服务器密钥文件 server.key 和服务器证书 server.crt：

```
#生成 CA 密钥
openssl genrsa -out ca.key 2048

#生成 CA 证书，days 参数以天为单位设置证书的有效期。在本过程中会要求输入证书的所在地、公司名、站点名等
openssl req -x509 -new -nodes -key ca.key -days 365 -out ca.crt
```

```
#生成服务器证书 RSA 的密钥对
openssl genrsa -out server.key 2048

#生成服务器端证书 CSR，本过程中会要求输入证书所在地、公司名、站点名等。
openssl req -new -key server.key -out server.csr

#生成服务器端证书 ca.crt
openssl x509 -req -in server.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out server.crt
-days 365
```

上述命令生成服务器端证书时，必须在 Common Name (CN) 字段中如实输入站点的访问地址。即如果站点通过 `www.mysite.com` 访问，则必须定义 `CN=www.mysite.com`；如果通过 IP 地址访问，则需设置 CN 为具体的 IP 地址。

### 3. 配置 Nginx HTTPS 服务器

在站点配置文件 `/etc/nginx/sites-enabled/default` 中添加如下 `server` 段，可以定义一个基于 HTTPS 的接口，该接口的服务器端程序仍旧为 uWSGI 接口 `127.0.0.1:3011`。

```
server {
    listen      443;                                #HTTPS 服务端口
    server_name 0.0.0.0;                             #本机上的所有 IP 地址
    ssl         on;
    ssl_certificate    /etc/nginx/ssl/server.crt;
    ssl_certificate_key /etc/nginx/ssl/server.key;

    location \ {
        uwsgi_pass http://127.0.0.1:3011;
    }
}
```

其中需要注意的是参数 `ssl_certificate` 和 `ssl_certificate_key` 需要分别指定生成的服务器证书和服务器密钥的全路径文件名。

至此，我们已经可以使用浏览器访问服务器的 443 端口进行 HTTPS 加密通信了。

## 5.4 本章总结

---

对本章内容总结如下。

- 讲解什么是网络框架，以及网络框架常用的 MVC 架构。
- 讲解 4 种 Python 网络开发框架：Django、Tornado、Flask、Twisted。
- 讲解 Python 虚环境概念，使用 Python 虚环境隔离不同的开发环境。
- 讲解 easy\_install 和 pip 技术，使用它们快速安装和管理 Python 组件。
- 讲解 WSGI 与 Python 网络程序的关系，以及 Nginx+uWSGI 的站点配置方法。
- 使用 OpenSSL 命令管理服务器证书，并使用 Nginx 配置安全的 HTTPS 站点。