

## 第 5 章



# Web 框架基础

由于 Python 简单易懂，可维护性强，所以越来越多的互联网公司使用 Python 进行 Web 开发，如豆瓣、知乎等网站。本章将介绍 Web 框架基础、常用的 Python Web 框架、开发环境准备以及 Web 框架的云服务部署等内容。

## 5.1 Web 框架简介

### 5.1.1 什么是 Web 框架

Web 框架是用来简化 Web 开发的软件框架。事实上，框架并不是什么新技术，它只是一些能够实现常用功能的 Python 文件。可以把框架看作是一系列工具的集合，其存在是为了避免重新发明“轮子”，以在创建新项目时减少开发成本。

一个典型的框架，通常会提供如下常用功能。

- 管理路由。
- 支持数据库。
- 支持 MVC。
- 支持 ORM。
- 支持模板引擎。
- 管理会话和 Cookies。

### 5.1.2 什么是 MVC

MVC (Model View Controller) 早在 1978 年就作为 Smalltalk 的一种设计模式被提出来，并应用到了 Web 应用上。Model (模型) 用于封装与业务逻辑相关的数据和数据处理方法，View (视图) 是数据的 HTML 展现，Controller (控制器) 负责响应请求，协调 Model 和 View。将 Model、View 和 Controller 分开，是一种典型的关注点分离的思想，不仅使代码复用性和组织性更好，还使得 Web 应用的配置性和灵活性更好。常见的 MVC 模式如图 5.1 所示。

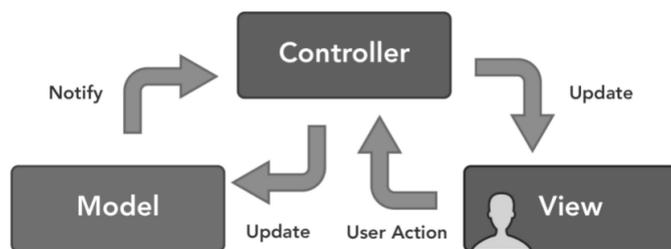


图 5.1 MVC 模式示意图

### 5.1.3 什么是 ORM

4.9.1 节中介绍过，ORM 是随着面向对象的软件开发方法发展而产生的。面向对象的开发方法是当今企业级应用开发环境中的主流开发方法，关系型数据库是企业级应用环境中永久存放数据的主流数据存储系统。对象和关系数据是业务实体的两种表现形式，业务实体在内存中表现为对象，在数据库中表现为关系数据。内存中的对象之间存在关联和继承关系，而在数据库中，关系数据无法直接表达多对多关联和继承关系。因此，ORM 系统一般以中间件的形式存在，主要实现程序对象到关系型数据库数据的映射。ORM 与数据库的对应关系如图 5.2 所示。

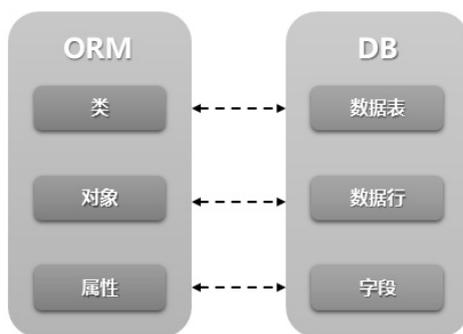


图 5.2 ORM 与数据库的对应关系

### 5.1.4 什么是模板引擎

模板引擎是为了使用户界面与业务数据（内容）分离而产生的，它可以生成特定格式的文档，用于网站的模板引擎一般生成一个标准的 HTML 文档。Python 很多 Web 框架都内置了模板引擎，使用了模板引擎可以在 HTML 页面中使用变量，例如：

```
01 <html>
02 <head>
03 <title>{{title}}</title>
04 </head>
05 <body>
06 <h1>Hello,{{username}}!</h1>
```

```
07 </body>
08 </html>
```

上述代码中的`{{}}`变量会被替换成变量值，这就可以让程序实现界面与数据相分离，业务代码与逻辑代码相分离，从而大大提升开发效率，良好的设计也使得代码重用变得更加容易。

## 5.2 常用的 Python Web 框架

第 3 章中我们学习了 WSGI（服务器网关接口），它是 Web 服务器和 Web 应用程序或框架之间的一种简单而通用的接口。也就是说，只要遵循 WSGI 接口规则，就可以自主开发 Web 框架。市面上现存的各种开源 Web 框架至少有上百个，关于 Python 框架优劣的讨论也仍在继续。作为初学者，应该选择一些主流的框架来学习使用。这是因为主流框架文档齐全，技术积累较多，社区繁盛，并且能得到更好的支持。下面介绍几种主流的 Python Web 框架。

### 1. Django

这可能是最广为人知、使用也最广泛的 Python Web 框架了。Django 拥有世界上最大的社区，最多的包。它的文档非常完善，并且提供了一站式的解决方案，包括缓存、ORM、管理后台、验证、表单处理等，使得开发复杂的由数据库驱动的网站变得简单。但是，Django 系统耦合度较高，替换掉内置的功能比较麻烦，所以学习曲线也相当陡峭。

### 2. Flask

Flask 是一个轻量级 Web 应用框架。它的名字暗示了它的含义，基本上就是一个微型的胶水框架。它把 Werkzeug 和 Jinja 粘合在了一起，所以很容易被扩展。Flask 有许多的扩展可以使用，同时也有—群忠诚的粉丝和不断增加的用户群。它有一份很完善的文档，甚至还有一份唾手可得的常见范例。Flask 很容易使用，只需要几行代码就可以写出来一个“HelloWorld”。

### 3. Tornado

Tornado 不单单是个框架，还是个 Web 服务器。它一开始是给 FriendFeed 开发的，2009 年时开始给 Facebook 使用。它是为了解决实时服务而诞生的。为了做到这一点，Tornado 使用了异步非阻塞 IO，所以它的运行速度非常快。

### 4. FastAPI

FastAPI 是一个现代的快速（高性能）Python Web 框架。它基于标准的 Python 类型提示，使用 Python 5.6+ 构建 API。FastAPI 使用了类型提示，能够减少开发人员容易引发的错误。此外，FastAPI 可以自动生成 API 文档，编写 API 接口后，可以使用符合标准的 UI 如 Swagger UI、ReDoc 等来使用 API。

以上 4 种框架各有优劣，使用时需要根据自身的应用场景选择适合自己的 Web 框架。在后面的章节中，会详细介绍每一个框架的使用方法。

## 5.3 准备开发环境

### 5.3.1 创建虚拟环境

#### 1. 为什么要使用虚拟环境

创建项目时，经常会用到第三方包和模块，且这些包和模块会随时间的增加而变更版本。例如，我们在创建第 1 个应用程序时，使用的框架是 Django 1.0。当开发第 2 个应用程序时，Django 版本已经升级到 2.0，这意味着一个 Python 安装可能无法满足每个应用程序的要求。这就导致需求存在冲突，无论安装版本 1.0 或 2.0，都将导致某一个应用程序无法运行。

如何解决这种问题呢？Python 提供的解决方案就是创建多个虚拟环境（Virtual Environment）。一个虚拟环境就是一个目录树，其中安装有特定的 Python 版本，以及许多其他包。

对于不同的应用可以使用不同的虚拟环境，这样就可以解决需求相冲突的情况。例如，应用程序 A 使用安装了 1.0 版本的虚拟环境，而应用程序 B 使用安装了 2.0 版本的另一个虚拟环境。如果应用程序 B 要求将某个库升级到 5.0 版本，也不会影响应用程序 A 的环境。多个虚拟环境的使用如图 5.3 所示。

#### 2. 安装 Virtualenv

Virtualenv 的安装非常简单，可以使用如下命令进行安装。

```
pip install virtualenv
```

安装完成后，可以使用如下命令检测 Virtualenv 版本。

```
virtualenv --version
```

如果运行效果如图 5.4 所示，则说明安装成功。



图 5.3 安装多个虚拟环境

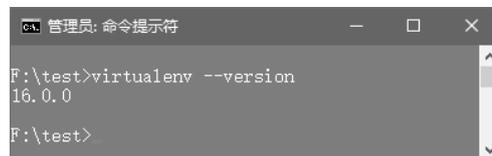


图 5.4 查看 Virtualenv 版本

#### 3. 创建虚拟环境

下一步是使用 Virtualenv 命令创建 Python 虚拟环境。这个命令只有一个必需的参数，即虚拟环境

的名字。按照惯例，一般虚拟环境会被命名为 `venv`。运行如下命令，如图 5.5 所示。

```
virtualenv venv
```

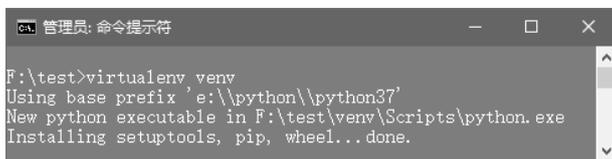


图 5.5 创建 `venv` 虚拟环境

运行完成后，在运行的目录下会新增一个 `venv` 文件夹，保存了一个全新的虚拟环境，目录结构如图 5.6 所示。

#### 4. 激活和关闭虚拟环境

在使用这个虚拟环境之前，需要先将其激活。不同的操作系统，激活 `venv` 虚拟环境的命令不同。Windows 系统下激活虚拟环境的命令如下：

```
venv\Scripts\activate
```

MacOS 或 Linux 系统下激活虚拟环境的命令如下：

```
source venv/bin/activate
```

激活成功后，会在命令行提示符前面新增 “(venv)” 标志，如图 5.7 所示。

使用完成后，可以使用 `deactivate` 命令关闭虚拟环境，如图 5.8 所示。

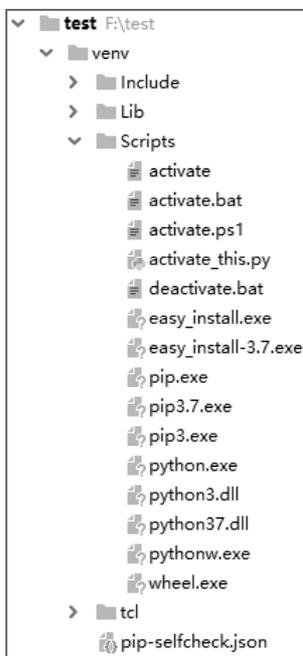


图 5.6 创建虚拟环境



图 5.7 激活虚拟环境后效果 (1)



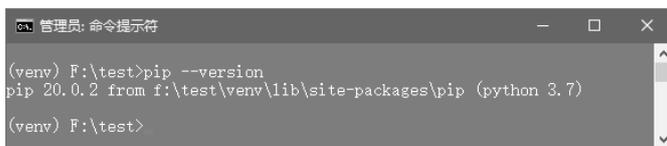
图 5.8 激活虚拟环境后效果 (2)

## 5.3.2 使用 pip 包管理工具

Web 开发过程中，除了可以使用 Python 内置的标准模块外，还需要使用很多的第三方模块。Python 提供了 pip 工具用来下载和管理第三方包。可以使用如下命令来检测是否可以使用 pip 工具。

```
pip --version
```

运行结果如图 5.9 所示，则表示可以使用 pip 工具。



```
管理员: 命令提示符
(venv) F:\test>pip --version
pip 20.0.2 from f:\test\venv\lib\site-packages\pip (python 3.7)
(venv) F:\test>
```

图 5.9 查看 pip 版本

### 1. 安装包

pip 使用如下命令安装包。

```
pip install 包名
```

例如，使用 pip 安装 beautifultable 模块，如图 5.10 所示。



```
管理员: 命令提示符
(venv) F:\test>pip install beautifultable
Looking in indexes: https://pypi.douban.com/simple
Collecting beautifultable
  Downloading https://pypi.doubanio.com/packages/d9/56/eaf1b9f2b323e05dce573f88c72eaa0107610db709b8bee97b776903ac55/beautifultable-0.8.0-py2.py3-none-any.whl (22 kB)
Installing collected packages: beautifultable
Successfully installed beautifultable-0.8.0
(venv) F:\test>
```

图 5.10 安装 beautifultable 模块

此外，pip 也可以安装指定版本的包，命令如下：

```
pip install 包名==版本号
```

例如，安装 0.8.0 版本的 beautifultable，命令如下：

```
pip install beautifultable==0.8.0
```



#### 说明

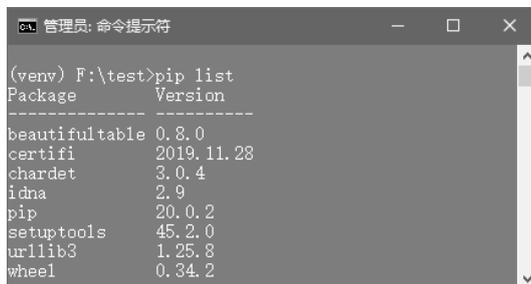
在虚拟环境下安装的包只能在该虚拟环境下使用，在全局环境或其他虚拟环境下无法使用。

### 2. 显示全部安装包

pip 使用如下命令显示已经安装的全部包名及版本号。

```
pip list
```

显示效果如图 5.11 所示。



```
管理员: 命令提示符
(venv) F:\test>pip list
Package      Version
-----
beautifultable 0.8.0
certifi      2019.11.28
chardet      3.0.4
idna         2.9
pip          20.0.2
setuptools   45.2.0
urllib3      1.25.8
wheel        0.34.2
```

图 5.11 显示全部已经安装的包

此外，还可以使用如下命令查看可以升级的包。

```
pip list --outdate
```

### 3. 升级包

使用如下命令升级包。

```
pip install --upgrade 包名
```

以上命令可以升级到最新版的包，也可以通过使用==、>=、<=、>、<将包升级到指定版本号。

### 4. 卸载包

使用如下命令卸载包。

```
pip uninstall 包名
```

### 5. 以 requirements 参数的格式输出包

将一个已经开发完成的项目迁移到另一个全新的 Python 环境时，可以将原项目中的包逐一安装到新环境中，但显然这种方式比较烦琐，而且容易遗漏。此时，可以使用如下方法解决环境迁移的问题。

(1) 使用如下命令将已经安装好的包输出到 requirements.txt 文件中。

```
pip freeze > requirements.txt
```



#### 说明

上述命令中，“>requirements.txt”表示输出到 requirements.txt 文本文件中。输出的文件名字可以自定义。根据惯例，通常使用 requirements.txt。

requirements.txt 文件中包含了包名以及版本号，例如：

```
PackageVersion
-----
certifi2018.11.29
chardet5.0.4
pip20.0.2
```

```
pygame1.9.6  
PyMySQL0.8.0
```

(2) 在全新的 Python 环境下，一次安装 requirements.txt 文件中的所有包，命令如下：

```
pip install -r requirements.txt
```

### 5.3.3 使用国内镜像源加速下载

在使用 pip 下载安装第三方包时，经常会因为下载超时而报错。这是由于下载包的服务器在国外，所以会出现访问超时的情况。可以使用国内镜像源来解决此类问题，比较常用的国内镜像源有 3 个。

- 清华大学：<https://pypi.tuna.tsinghua.edu.cn/simple>
- 阿里云：<http://mirrors.aliyun.com/pypi/simple/>
- 豆瓣：<http://pypi.douban.com/simple/>

使用镜像源的方式有两种：临时使用和默认永久使用。

#### 1. 临时使用

临时使用指的是每次安装包时设置一次，下次再安装新的包时还需要再设置。例如，临时使用清华大学镜像源安装 beautifultable，命令如下：

```
pip install -i https://pypi.tuna.tsinghua.edu.cn/simple beautifultable
```



#### 说明

上述命令中，“-i”参数是 index 的缩写，表示索引，后面紧接着是镜像源的地址。

#### 2. 默认使用

如果感觉临时使用镜像源的方式比较烦琐，可以将镜像源设置成配置文件，当使用 pip 下载包时，默认执行该配置文件，到指定镜像源中取下载包。以配置清华大学镜像源为例，配置信息如下：

```
[global]  
index-url=https://pypi.tuna.tsinghua.edu.cn/simple  
[install]  
trusted-host=mirrors.aliyun.com
```

对于不同的操作系统，配置文件所在的路径并不相同。

Windows 系统下，在 user 目录中创建一个 pip 目录，如 C:\Users\Administrator\pip，然后在 pip 文件夹下新建一个 pip.ini 文件，在 pip.ini 文件中添加清华大学镜像源的配置。



#### 注意

Administrator 是默认的用户名，读者需要根据自己电脑具体情况自行替换。

Linux 系统或 MacOS 系统下，创建 ~/.config/pip/pip.conf 目录，并在 pip.conf 文件中添加清华大学镜像源的配置。

## 5.4 部署腾讯云服务器

### 5.4.1 WSGI+Gunicorn+Nginx+Supervisor 部署方式

Flask、Django 等框架自身都包含 Web 服务器，在本地开发时可以直接使用内置的服务器来启动项目。由于性能问题，框架自带的 Web 服务器主要用于开发测试，当项目线上发布时，还需要使用高性能的 WSGIServer。下面介绍一下部署 Python Web 项目时的一些基本概念。

#### 1. WSGI

WSGI 中存在两种角色：接受请求的 Server（服务器）和处理请求的 Application（应用）。当 Server 收到一个请求后，可以通过 Socket 把环境变量和一个 Callback 回调函数传给后端 Application，Application 在完成页面组装后通过 Callback 把内容返回给 Server，最后 Sever 再将响应返回给 Client（客户端）。

#### 2. Gunicorn

常用的 WSGI Server 容器有 Gunicorn 和 uWSGI。Gunicorn 直接用命令启动，不需要编写配置文件，相对 uWSGI 要容易很多，所以本节使用 Gunicorn 作为容器。

Gunicorn（Green Unicorn）是从 Ruby 社区的 Unicorn 移植到 Python 上的一个 WSGI HTTP Server。Gunicorn 使用 pre-fork worker 模型，Gunicorn 服务器与各种 Web 框架广泛兼容，实现简单，服务器资源少且速度更快。

#### 3. Nginx

通常在 Gunicorn 服务前再部署一个 Nginx 服务器。Nginx 是一个 Web 服务器，是一个反向代理工具，通常用它来部署静态文件。既然通过 Gunicorn 已经可以启动服务了，那为什么还要添加一个 Nginx 服务呢？

Nginx 作为一个 HTTP 服务器，它有很多 uWSGI 不具备的特性。

- ☑ 静态文件支持。经过配置之后，Nginx 可以直接处理静态文件请求而不用经过应用服务器，避免占用宝贵的运算资源；还能缓存静态资源，使访问静态资源的速度提高。
- ☑ 抗并发压力。Nginx 可以吸收一些瞬时的高并发请求，先保持住连接（缓存 HTTP 请求），然后在后端慢慢处理。如果让 Gunicorn 直接提供服务，浏览器发起一个请求，鉴于浏览器和网络情况都是未知的，HTTP 请求的发起过程可能比较慢，而 Gunicorn 只能等待请求发起完成后，才去真正处理请求，处理完成后，等客户端完全接收请求后，才继续下一个。
- ☑ HTTP 请求缓存头处理得也比 Gunicorn 和 uWSGI 完善。
- ☑ 多台服务器时，可以提供负载均衡和反向代理。

#### 4. Supervisor

当程序异常退出时，我们希望进程重新启动。Supervisor 是一个进程管理工具，使用 Supervisor 看

守进程，一旦异常退出，它会立即启动进程。

综上所述，框架部署的链路一般是：Nginx→WSGI Server→Python Web 程序，通常还会结合 Supervisor 工具来监听启停，如图 5.12 所示。

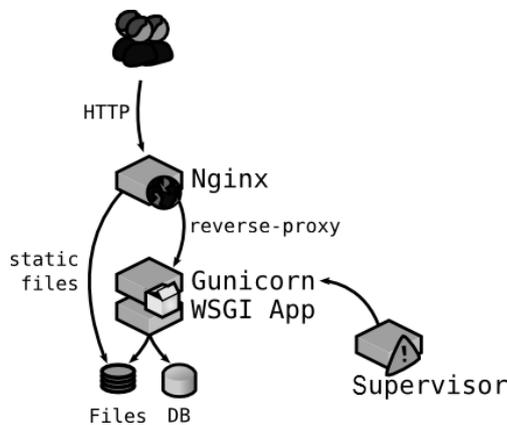


图 5.12 框架部署链路

## 5.4.2 常用的云服务器

本地开发的项目只能通过局域网在本地进行访问，为了能够让更多的人通过互联网访问到项目，需要购买服务器，并将项目部署到服务器中。

近年来，云服务器在中国快速普及开来。之前，如果想要搭建一个网站，就要购买服务器或者合租服务器，又或者购买一些虚拟主机，看个人选择而不同。现在搭建一个网站，只需在网上选择一个云服务器厂商，按照自己的配置需求点几下，就拥有了自己的云服务器，计费方式可按照需求包月或包年等。相比传统的购买服务器，既节省了经济成本，又节约了大量时间。

云服务器厂商多如牛毛，如阿里云、腾讯云、华为云等，读者可以根据自身情况进行选择。本节主要介绍如何在腾讯云服务器上部署 Python Web 项目，其他云服务器的部署方式大同小异。

### 1. 注册腾讯云账号

进入腾讯云官方网址 <https://cloud.tencent.com>，可以通过微信或 QQ 进行注册。

### 2. 购买 Linux 云服务器

在腾讯云首页单击“云服务器”选项，再单击“立即选购”按钮进入云服务购买页面，读者需要结合自身情况选择如下选项。

- 地域：选择最近的一个地区，例如“北京”。
- 机型：选择需要的云服务器机型配置，这里选择“入门设置（1核 1GB）”。
- 镜像：选择需要的云服务器操作系统，这里选择“Ubuntu Server 16.04.1 LTS 64 位”。
- 公网带宽：选中该复选框后，系统会为你分配公网 IP，默认为 1Mbps，可以根据需求调整。
- 购买数量：默认为“1 台”。
- 购买时长：默认为“1 个月”。

购买云服务器配置页面如图 5.13 所示。

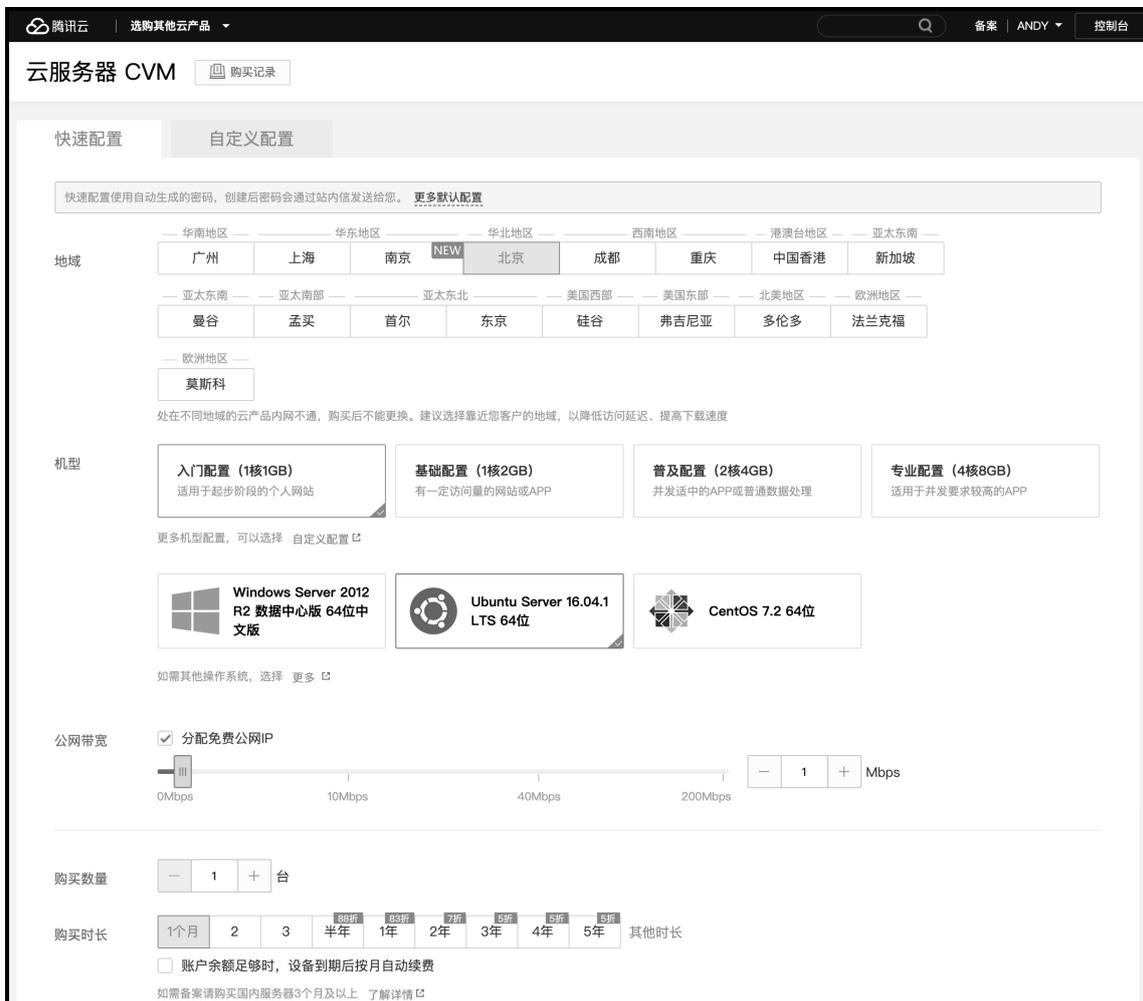


图 5.13 云服务器配置页面

付费完成后，即完成了云服务器的购买。接下来云服务器可以作为个人虚拟机或者建站的服务器。

### 3. 登录云服务器

云服务器创建成功后，系统将为用户分配云服务器登录密码并发送到站内信中，如图 5.14 所示。



图 5.14 获取初始密码

登录云服务器控制台，在“实例”列表中找到刚购买的云服务器，如图 5.15 所示。在右侧操作栏中单击“登录”按钮，进入登录 Linux 实例页面，选择标准登录方式，单击“立即登录”按钮，进入登录页面，如图 5.16 所示。



图 5.15 控制台实例



图 5.16 Linux 实例登录页面

在登录页面，默认的用户名是 `ubuntu`，输入初始密码，单击“确定”按钮，进入终端页面，如图 5.17 所示。

#### 4. 重置实例密码

如果遗忘了密码，可以在控制台上重新设置实例的登录密码。在实例的管理页面，选择需要重置密码的云服务器行，选择“更多”→“密码/密钥”→“重置密码”命令，如图 5.18 所示。

```

清理终端

* Socket connection established *
Welcome to Ubuntu 18.04.4 LTS (GNU/Linux 4.15.0-88-generic x86_64)

* Documentation:  https://help.ubuntu.com
* Management:    https://landscape.canonical.com
* Support:       https://ubuntu.com/advantage

System information as of Mon May  4 11:12:32 CST 2020

System load:  0.03          Processes:      114
Usage of /:   6.9% of 49.15GB Users logged in:  1
Memory usage: 20%         IP address for eth0: 172.16.0.9
Swap usage:   0%

* Ubuntu 20.04 LTS is out, raising the bar on performance, security,
and optimisation for Intel, AMD, Nvidia, ARM64 and Z15 as well as
AWS, Azure and Google Cloud.

https://ubuntu.com/blog/ubuntu-20-04-lts-arrives

* Canonical Livepatch is available for installation.
- Reduce system reboots and improve kernel security. Activate at:
https://ubuntu.com/livepatch

Last login: Sun May  3 16:53:50 2020 from 119.28.22.215
ubuntu@VM-0-9-ubuntu:~$

```

图 5.17 终端页面



图 5.18 重置密码

### 5.4.3 安装 pip 包管理工具

输入正确的用户名和密码进入终端后，可以通过如下指令查看当前系统的版本号。

```

ubuntu@VM-0-9-ubuntu:~$ cat/etc/issue
Ubuntu 18.04.4 LTS \n \l

```



#### 说明

第 1 行“\$”后面的是命令内容，第 2 行是输出结果。

Ubuntu 18.04.4 版本自带了 Python 2 和 Python 3，可以通过如下指令查看 Python 版本。

```

ubuntu@VM-0-9-ubuntu:~$ python --version
Python2.7.15+
ubuntu@VM-0-9-ubuntu:~$ python3 --version
Python3.6.9

```

运行结果如图 5.19 所示。

```

ubuntu@VM-0-9-ubuntu:~$ cat /etc/issue
Ubuntu 18.04.4 LTS \n \l

ubuntu@VM-0-9-ubuntu:~$ python --version
Python 2.7.15+
ubuntu@VM-0-9-ubuntu:~$ python3 --version
Python 3.6.9
ubuntu@VM-0-9-ubuntu:~$

```

图 5.19 查看 Python 版本

当前有 Python 2 和 Python 3 两个版本，它们分别对应着 pip 和 pip3。我们要使用 Python 3 版本，在终端输入如下命令：

```
ubuntu@VM-0-9-ubuntu:~$ pip3
Command 'pip3' not found , but can be installed with:
sudo apt install python3 -pip
```

pip3 命令不存在，可以安装提示来安装 pip3，运行如下命令：

```
ubuntu@VM-0-9-ubuntu:~$ sudo apt install python3 -pip
```

安装完成后，输入如下命令：

```
ubuntu@VM-0-9-ubuntu:~$ pip3 --version
pip 9.0.1 from /usr/lib/python3/dist-packages(python5.6)
```

此时，说明已经成功安装了 Python 3 版本的包管理工具。

#### 5.4.4 安装虚拟环境

接下来安装 virtualenv 虚拟环境，使用如下命令：

```
sudo pip3 install virtualenv
```



##### 说明

由于默认使用的登录账号是 ubuntu，安装 virtualenv 时，会提示 Permission Error 权限不足。此时在命令前添加 sudo，表示以系统管理者的身份执行指令，也就是说，经由 sudo 执行的指令就好像是 root 亲自执行。

接下来，选定创建项目目录，在 /var/www/html 目录下创建 flask\_test 文件夹，命令如下：

```
ubuntu@VM-0-9-ubuntu:~$ sudo mkdir /var/www/html/flask_test
```

修改 flask\_test 文件的所有者为 ubuntu 用户，命令如下：

```
ubuntu@VM-0-9-ubuntu:~$ sudo chown -R ubuntu/var/www/html/flask_test
```

接下来，进入 flask\_test 目录，并创建虚拟环境，命令如下：

```
ubuntu@VM-0-9-ubuntu:~$ cd /var/www/html/flask_test/
ubuntu@VM-0-9-ubuntu:/var/www/html/flask_test$ virtualenv -p python3 venv
Already using interpreter /usr/bin/python3
Using base prefix '/usr'
New python executable in /var/www/html/flask_test/venv3/bin/python3
Also creating executable in /var/www/html/flask_test/venv3/bin/python
Please make sure you remove any previous custom paths you're your /home/ubuntu/.pydistutils.cfgfile.
Installing setup tools, pkg_resources, pip, wheel...done.
```

**说明**

安装虚拟环境时使用命令“`virtualenv -p python3 venv`”，其中 `-p` 参数用于指定 Python 3 版本，否则默认为 Python 2。

安装完成后，激活虚拟环境，命令如下：

```
ubuntu@VM-0-9-ubuntu:/var/www/html/flask_test$ source venv/bin/activate
(venv)ubuntu@VM-0-9-ubuntu:/var/www/html/flask_test$
```

在虚拟环境下查看 pip 版本，命令如下：

```
(venv)ubuntu@VM-0-9-ubuntu:/var/www/html/flask_test$ pip --version
pip 20.1 from /var/www/html/flask_test/venv/lib/python5.6/site-packages/pip(python5.6)
```

接下来，需要安装一个 Python Web 框架。Flask 框架比较简单，因此在虚拟环境中安装 Flask 框架，命令如下：

```
(venv)ubuntu@VM-0-9-ubuntu:/var/www/html/flask_test$ pip install flask
```

安装完成后，使用 vim 编辑器创建一个 `run.py` 文件，命令如下：

```
vim run.py
```

在打开的文件内，按下键盘中的 I 键，进入 vim 插入模式，输入如下代码：

```
01 from flask import Flask
02
03
04 app=Flask(__name__)
05
06 @app.route("/")
07 def index():
08     return 'hello world'
09
10 if __name__=="__main__":
11     app.run()
```

输入完成，按 Esc 键，切换到底线命令模式，在最底一行输入“`:.wq`”，按下 Enter 键，保存并退出。在虚拟环境下，输入如下命令运行程序。

```
(venv)ubuntu@VM-0-9-ubuntu:/var/www/html/flask_test$ python run.py
*Serving Flask app "run"(lazy loading)
*Environment: production
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
*Debug mode: off
*Running on http://127.0.0.1:5000/(Press CTRL+C to quit)
```

此时我们使用的是 Flask 内置的服务器，只能通过本地进行访问。5.4.5 节将介绍如何使用 Gunicorn

服务启动。



#### 说明

读者需要了解最基本的 vim 知识。

## 5.4.5 安装 Gunicorn

Gunicorn 是使用 Python 开发的,因此可以直接使用 pip 进行安装。在 venv 虚拟环境下安装 Gunicorn 的命令如下:

```
(venv)ubuntu@VM-0-9-ubuntu:/var/www/html/flask_test$ pip install gunicorn
```

安装成功以后,可以通过以下两种方式来启动服务。

### 1. 添加参数启动服务

可通过如下命令直接启动 Gunicorn。

```
(venv)ubuntu@VM-0-9-ubuntu:/var/www/html/flask_test$ gunicorn -w 3 -b 0.0.0.0:9100 run:app
```

参数说明如下。

- w: 用于处理工作的进程数量。
- b: 绑定运行的主机和端口。
- run: 执行的 Python 文件 run.py。
- app: Flask APP 应用名称。

启动后运行效果如下:

```
[2020-05-04 17:49:27+0800][27943][INFO]Starting gunicorn 20.0.4
[2020-05-04 17:49:27+0800][27943][INFO]Listenin gat:http://0.0.0.0:9100(27943)
[2020-05-04 17:49:27+0800][27943][INFO]Using worker:sync
[2020-05-04 17:49:27+0800][27946][INFO]Booting worker with pid:27946
[2020-05-04 17:49:27+0800][27947][INFO]Booting worker with pid:27947
[2020-05-04 17:49:27+0800][27948][INFO]Booting worker with pid:27948
```

此时,可以在浏览器中输入公网 IP 地址来访问 Flask 项目,运行结果如图 5.20 所示。



图 5.20 访问公网 IP 地址

此外, Gunicorn 还有很多常用的启动参数,如表 5.1 所示。

表 5.1 Gunicorn 常用启动参数及说明

参 数	说 明
-c CONFIG, --config=CONFIG	指定配置文件
-b BIND, --bind=BIND	绑定运行的主机加端口
-w INT, --workers INT	用于处理工作进程的数量。整数，默认为 1
-k STRTING, --worker-class STRTING	要使用的工作模式，默认为 sync 异步。类型可以是 sync、eventlet、gevent、tornado、gthread、gaihttp
--threads INT	处理请求的工作线程数，使用指定数量的线程运行每个 worker。为正整数，默认为 1
--worker-connections INT	最大客户端并发数量，默认为 1000
--backlog INT	等待连接的最大数，默认为 2048
-p FILE, --pid FILE	设置 pid 文件的文件名，如果不设置，将不会创建 pid 文件
--access-logfile FILE	日志文件路径
--access-logformat STRING	日志格式，--access_log_format'%(h)s%(l)s%(u)s%(t)s'
--error-logfile FILE, --log-file FILE	错误日志文件路径
--log-level LEVEL	日志输出等级
--limit-request-line INT	限制 HTTP 请求行的允许大小，默认为 4094。取值范围为 0~8190，此参数可以防止任何 DDOS 攻击
--limit-request-fields INT	限制 HTTP 请求头字段的数量，以防止 DDOS 攻击。与 limit-request-field-size 一起使用，可以提高安全性。默认为 100，最大值为 32768
--limit-request-field-size INT	限制 HTTP 请求中请求头的大小，默认为 8190。值是一个整数或者 0，当该值为 0 时，表示不对请求头大小做限制
-t INT, --timeout INT	超过设置后，工作将被杀掉并重新启动，默认为 30s，Nginx 默认为 60s
--reload	在代码改变时自动重启，默认为 False
--daemon	是否以守护进程启动，默认为 False
--chdir	在加载应用程序之前切换目录
--graceful-timeout INT	默认为 30，超时（从接收到重启信号开始）之后仍然活着的工作将被强行杀死。一般采用默认值
--keep-alive INT	在 keep-alive 连接上等待请求的秒数，默认情况下值为 2。一般设定为 1~5s
--spew	打印服务器执行过的每一条语句，默认为 False。此选择为原子性的，即要么全部打印，要么全部不打印
--check-config	显示当前的配置，默认为 False，即显示
-e ENV, --env ENV	设置环境变量

## 2. 加载配件文件启动服务

如果启动 Gunicorn 时加载的参数很多，那么，第一种直接启动的方式就不再适用了，此时可以使用加载配置文件的方式来启动 Gunicorn。

在 flask\_test 文件夹下创建 gunicorn 文件夹，命令如下：

```
mkdir /var/www/html/flask_test/gunicorn
```

然后使用 `cd` 命令进入该目录，命令如下：

```
ubuntu@VM-0-9-ubuntu:~$ cd /var/www/html/flask_test/gunicorn
```

使用 `vim` 编写 `gunicorn_conf.py` 文件，命令如下：

```
vim gunicorn_conf.py
```

`gunicorn_conf.py` 文件的关键代码如下：

```
01 import multiprocessing
02
03
04 bind='0.0.0.0:9100'
05 workers=multiprocessing.cpu_count()*2+1           #进程数
06 reload=True
07 loglevel='info'
08 timeout=600
09
10 log_path="/tmp/logs/flask_test"
11 accesslog=log_path+'/gunicorn.access.log'
12 errorlog=log_path+'/gunicorn.error.log'
```

上述代码中的参数可以参照表 5.1 中的常用启动参数及说明，其中，`log_path` 变量读者可以自行定义。启动 Gunicorn 出错时，可以查看 `errorlog` 错误日志。

接下来，先终止 Gunicorn 进程，命令如下：

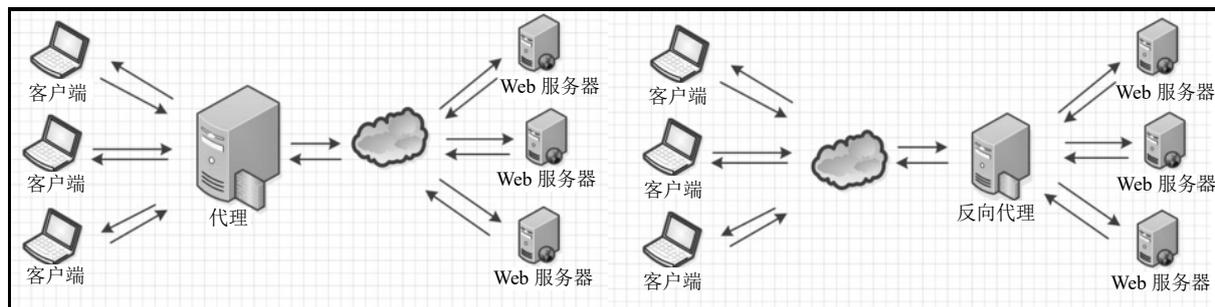
```
ubuntu@VM-0-9-ubuntu:/var/www/html/flask_test$ pkill gunicorn
```

然后在虚拟环境下以加载配置文件的方式启动 Gunicorn，命令如下：

```
(venv)ubuntu@VM-0-9-ubuntu:/var/www/html/flask_test$ gunicorn -c
gunicorn/gunicorn_conf.pyrun:app
[2020-05-0613:00:28+0800][27753][INFO]Starting gunicorn20.0.4
[2020-05-0613:00:28+0800][27753][INFO]Listenin gat:http://0.0.0.0:9100(27753)
[2020-05-0613:00:28+0800][27753][INFO]Using worker:sync
[2020-05-0613:00:28+0800][27756][INFO]Booting worker with pid:27756
[2020-05-0613:00:28+0800][27757][INFO]Booting worker with pid:27757
[2020-05-0613:00:28+0800][27758][INFO]Booting worker with pid:27758
```

## 5.4.6 安装 Nginx

Nginx 是一款轻量级的 Web 服务器和反向代理服务器，由于它的内存占用少，启动极快，高并发能力强，在互联网项目中受到广泛应用。通常在 Gunicorn 服务中添加一层 Nginx 反向代理。正向代理和反向代理如图 5.21 所示。



(a) 正向代理

(b) 反向代理

图 5.21 正向代理和反向代理

## 1. 安装 Nginx

在 Ubuntu 系统中使用如下命令安装 Nginx。

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

安装成功后 Nginx 会默认启动，此时在浏览器中访问公网 IP，运行结果如图 5.22 所示。



图 5.22 Nginx 启动成功

此外，Nginx 有 4 个主要的文件夹结构，目录及说明如下。

- ☑ /usr/sbin/nginx: 主程序。
- ☑ /etc/nginx: 存放配置文件。
- ☑ /usr/share/nginx: 存放静态文件。
- ☑ /var/log/nginx: 存放日志。

## 2. Nginx 基本命令

Nginx 启动之后，可以使用以下命令进行控制。

```
nginx -s <signal>
```

其中，-s 表示向主进程发送信号；signal 可以为以下 4 个参数。

- ☑ stop: 快速关闭。
- ☑ quit: 优雅关闭。
- ☑ reload: 重新加载配置文件。

☑ `reopen`: 重新打开日志文件。

当运行 `nginx -s quit` 时, Nginx 会等待工作进程处理完成当前请求, 然后将其关闭。当修改配置文件后, 并不会立即生效, 而是等待重启或者收到 `nginx -s reload` 信号。

当 Nginx 收到 `nginx -s reload` 信号后, 首先检查配置文件的语法。语法正确后, 主线程会开启新的工作线程, 并向旧的工作线程发送关闭信号。如果语法不正确, 则主线程回滚变化, 并继续使用旧的配置。当工作进程收到主进程的关闭信号后, 会在处理完当前请求之后退出。

### 3. Nginx 配置文件

Nginx 配置的核心是定义要处理的 URL 以及如何响应这些 URL 请求, 即定义一系列的虚拟服务器 (Virtual Servers), 控制对来自特定域名或者 IP 的请求的处理。

每一个虚拟服务器定义一系列的 `location`, 处理特定的 URI 集合。每个 `location` 都定义了对映射到自己请求的处理场景, 可以返回一个文件或者代理此请求。

Nginx 由不同的模块组成, 这些模块由配置文件中指定的指令控制。指令分为简单指令和块指令。一个简单指令包含指令名称和指令参数, 以空格分隔, 以分号 (;) 结尾。块指令与简单指令类似, 但是由大括号 (“{” 和 “}”) 包围。如果块指令的大括号中包含其他指令, 则称该指令为上下文 (如 `events`、`http`、`server` 和 `location`)。

配置文件中放在上下文之外的指令默认放在主配置文件中 (类似于继承主配置文件)。`events` 和 `http` 放置在主配置文件中, `server` 放置在 `http` 块指令中, `location` 放置在 `server` 块指令中。配置文件的注释以 “#” 开始。

### 4. 静态文件

Web 服务器的一个重要功能是服务静态文件 (如图像或静态 HTML 页面)。例如, Nginx 可以很方便地让服务器从 `/var/www/html` 获取 html 文件, 从 `/var/www/html/images` 获取图片来返回给客户端, 只需要在 `http` 块指令中的 `server` 块指令中设置两个 `location` 块指令即可。

首先, 进入 `/var/www/html` 目录, 在该目录下创建 `welcome.html`。然后再创建 `/data/images` 目录, 并将一些图片从本地上传至服务器。

接下来, 进入 `/etc/nginx/sites-enabled` 配置文件目录, 在该目录下的所有文件都会作为配置文件被加载进来。所以, 通常为网站单独创建一个配置文件。这里创建一个 `demo` 文件, 代码如下:

```
server {
    location / {
        root /var/www/html
    }
    location /images/ {
        root /var/www/html/images
    }
}
```

通常, 配置文件可以包括多个 `server` 块, 它们以端口和服务器名称进行区分。当 Nginx 决定某一个 `server` 处理请求后, 它将请求头中的 URI 和 `server` 块中的 `location` 块进行对比。加入 `location` 块指令到 `server` 中。

第一个 location 块指定 “/” 前缀与请求中的 URI 对比。对于匹配的请求，URI 将被添加到 root 指令中指定的路径，即/var/www/html，以此形成本地文件系统的路径。如访问 http://localhost/welcome.html，对应的服务器文件路径为/var/www/html/welcome.html。如果 URI 匹配多个 location 块，Nginx 采用最长前缀匹配原则（类似计算机网络里面的 IP 匹配），上面的 location 块前缀长度为 1，因此只有当所有其他 location 块匹配时，才使用该块。

例如，第二个 location 位置块，它将匹配以/images/（“/” 也匹配这样的请求，但具有较短的前缀）开始的请求。

配置完成后，使用如下命令重新加载 Nginx。

```
nginx -s reload
```

至此，已搭建好了一个可以正常运行的服务器，它监听 80 端口，并且可以在公网 IP 上访问。例如，访问公网 IP/welcome.html，运行结果如图 5.23 所示。访问公网 IP/images/qrcoder.jpg，运行结果如图 5.24 所示。

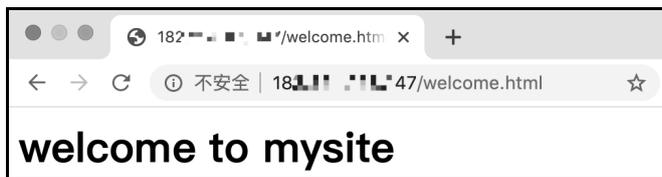


图 5.23 访问静态 HTML 文件



图 5.24 访问静态图片资源

## 5. 代理服务器

Nginx 的一个常见应用是将其设置为代理服务器（Proxy Server），即接受客户端的请求并将其转发给服务器，再接受服务器发来的响应，将它们发送到客户端。比如，我们可以用一个 Nginx 实例实现将对 8000 端口的请求，转发到服务器。

进入/etc/nginx/sites-enabled 配置文件目录，创建 flask\_demo 文件，代码如下：

```
server {  
    listen 8000;  
    listen [::]:8000;  
    server_name 182.254.165.147;  
    location / {  
        proxy_pass http://182.254.165.147:9100;  
        proxy_set_header Host      $host;  
        proxy_set_header X-Real-IP $remote_addr;  
    }  
}
```

上述代码中，设置监听 8000 端口，接受到请求后，通过 `proxy_pass` 设置代理转发至 9100 端口。参数说明如下。

- `listen`: 监听的端口。
- `server_name`: 监听地址。
- `proxy_pass`: 代理转发。
- `proxy_set_header`: 允许重新定义或添加字段传递给代理服务器的请求头。

重新加载 Nginx，在浏览器中访问公网 IP:8000，Nginx 会转发至公网 IP:9100，运行结果如图 5.25 所示。



图 5.25 代理转发效果

## 5.4.7 安装 Supervisor

Supervisor 是一个用 Python 编写的进程管理工具，它符合 C/S 架构体系，对应的角色分别为 Supervisorctl 和 SupervisorD。

- SupervisorD: 启动配置好的程序，响应 Supervisorctl 发过来的指令以及重启退出的子进程。
- Supervisorctl: 它以命令行的形式提供了一系列参数，以方便用户向 SupervisorD 发送指令，常用的有启动、暂停、移除、更新等命令。

使用 Ubuntu 系统命令或者 Python 包管理工具都可以安装 Supervisor。由于使用 Ubuntu 安装的 Supervisor 版本较低，所以推荐使用 pip 命令来安装。

为了方便查找路径，在 `/home/ubuntu` 目录下新建一个 `venv` 虚拟环境，命令如下：

```
ubuntu@VM-0-9-ubuntu:~$ virtualenv venv
```

创建完成后，激活该虚拟环境，并使用如下命令安装 Supervisor。

```
ubuntu@VM-0-9-ubuntu:~$ source venv/bin/activate
(venv) ubuntu@VM-0-9-ubuntu:~$ pip install supervisor
```

### 1. 创建配置文件

安装完 Supervisor 以后，在终端输入如下命令可以查看 supervisor 的基本配置。

```
(venv) ubuntu@VM-0-9-ubuntu:~$ echo_supervisord_conf
```

如果在终端看到输出配置文件内容，接下来在 `/etc/supervisor` 目录下创建 `supervisord.conf` 文件，命令如下：

```
(venv) ~$ sudo su - root -c "echo_supervisord_conf > /etc/supervisor/supervisord.conf"
```



## 说明

`su -root -c` 表示使用 `root` 用户权限执行命令。

执行完成后，在 `/etc/supervisor/` 目录下会生成一个 `supervisord.conf` 文件，使用 `vim` 编辑该文件，修改最后一行的代码。修改结果如下：

```
[include]
;files = relative/directory/*.ini
files = /etc/supervisor/conf.d/*.ini
```

上述代码的作用是将 `/etc/supervisor/conf.d` 目录下所有后缀为 `.ini` 的文件作为配置文件加载。

此外，默认文件将 `supervisord.pid` 以及 `supervisor.sock` 存放在 `/tmp` 目录下。注意，`/tmp` 目录是存放临时文件的，里面的文件会被 Linux 系统删除。一旦这些文件丢失，就无法再通过 `supervisorctl` 来执行相关命令了，而是会提示 `unix:///tmp/supervisor.sock` 不存在的错误。所以，需要将包含 `/tmp` 的目录做如下修改。

```
[supervisorctl]
serverurl=unix:///var/run/supervisor.sock ; use a unix:// URL  for a unix socket

[unix_http_server]
file=/var/run/supervisor.sock ; the path to the socket file

[supervisord]
logfile=/var/log/supervisord.log ; main log file; default $CWD/supervisord.log
pidfile=/var/run/supervisord.pid ; supervisord pidfile; default supervisord.pid
```

修改完成后，进入 `/etc/supervisor/conf.d` 目录，在该目录下使用 `vim` 编辑器创建 `test.ini` 配置文件。`test.ini` 文件代码如下：

```
[program:foo]
command=/bin/cat
```

接下来，使用如下命令启动 `supervisor`。

```
sudo supervisord -c /etc/supervisor/supervisor.conf
```

启动成功后，通过如下命令查看进程的状态。

```
(venv) ubuntu@VM-0-9-ubuntu:/etc/supervisor$ sudo supervisorctl status
foo                                RUNNING    pid 10133, uptime 0:19:53
```

## 2. Supervisorctl 常用命令

`supervisorctl status` 命令可以查看进程的状态。此外，`Supervisorctl` 还有很多其他常用的命令，如表 5.2 所示。

表 5.2 Supervisorctl 常用命令

参 数	说 明
status	查看进程状态
status <name>	查看<name>进程状态
start <name>	启动<name>进程
stop <name>	停止<name>进程
stop all	停止进程服务
restart <name>	重启<name>服务。注意，不会重新读取配置信息
restart all	重启全部服务。注意，不会重新读取配置信息
reload	重新启动远程监督者
reread	重新加载守护程序的配置文件，而无须添加/删除（不重新启动）
stop all	停止进程服务
add <name>	激活配置中进程/组的任何更新
remove <name>	从活动配置中删除进程/组
update	重新加载配置，并根据需要添加/删除，同时重新启动受影响的程序
tail	输出最新的 log 信息
shutdown	关闭 supervisor 服务

### 3. 配置 Gunicorn 启动程序

前面学习了如何使用 Gunicorn 来启动 Flask 程序，但如果 Gunicorn 服务器出现故障，Flask 程序就会中断。为了解决这个问题，可以使用 Supervisor 来监测 Gunicorn 进程。当 Gunicorn 服务停止时，令其自动重启。

首先需要在/etc/supervisor/conf.d 目录下新建一个 flask\_test 配置文件，配置如下：

```
[program:flask_test]
command=/var/www/html/flask_test/venv/bin/gunicorn -c gunicorn/gunicorn_conf.py run:app
directory=/var/www/html/flask_test
user=root
autostart=True
autorestart=True
startsecs=10
startretries=3
stdout_logfile=/var/log/flask_test_error.log
stderr_logfile=/var/log/flask_test_out.log
stopasgroup=True
stopsignal=QUIT
```

文件中参数说明如下。

- program: 程序名称。
- command: 要执行的命令。
- directory: 当 Supervisor 作为守护程序运行时，在守护程序之前 cd 到该目录。
- user: 以哪个用户执行。
- autostart: 是否与 Supervisor 一起启动。

- ☑ **autorestart**: 是否自动重启。
- ☑ **startsecs**: 延时启动时间，默认为 10 秒。
- ☑ **startretries**: 启动重试次数，默认为 3 次。
- ☑ **stdout\_logfile**: 正常输出日志。
- ☑ **stderr\_logfile**: 错误输出日志。
- ☑ **stopasgroup**: 如果为 `True`，则该标志使 Supervisor 将停止信号发送到整个过程组，并暗示 `killasgroup` 为 `True`。这对于程序（如调试模式下的 Flask）非常有用，这些程序不会将停止信号传播到其子级，而使它们成为孤立状态。
- ☑ **stopsignal**: 停止信号。

配置完成后，使用如下命令重启 Supervisor。

```
(venv) ubuntu@VM-0-9-ubuntu:~$ sudo supervisorctl reload
Restarted supervisord
```

重启后通过如下命令查看所有进程的状态。

```
(venv) ubuntu@VM-0-9-ubuntu:~$ sudo supervisorctl status
flask_test          RUNNING   pid 30683, uptime 0:00:41
foo                 RUNNING   pid 30684, uptime 0:00:41
```

为了验证 Supervisor 是否能够自动重启 Gunicorn，使用如下命令关闭 Gunicorn 进程。

```
(venv) ubuntu@VM-0-9-ubuntu:~$ sudo pkill gunicorn
```

在浏览器中访问公网 IP:9100 端口，发现 Flask 程序依然可以正常访问。此外，也可以通过如下命令对比 gunicorn 关闭前后进程 ID 是否发生变化。

```
ps aux | grep gunicorn
```

## 5.5 小 结

本章主要介绍 Web 框架相关的基础知识，首先介绍什么是 Web 框架以及 Web 框架需要具备哪些常用的功能。接下来，介绍 Python 中 4 个流行的 Web 框架，包括它们的特点及应用场景。最后介绍准备开发环境和部署到云服务器。其中，将项目部署到云服务器的内容是本章的难点，读者在学习的过程中可以先练习在本地测试开发，最后再来学习如何部署到云服务器上。