Django 项目部署指南

一、引言

本指南详细阐述了在 Ubuntu 服务器上运用 Nginx + WSGI 来部署 Django 项目的全流程，包含服务器搭建、数据库安装与配置、本地代码上传以及远程服务器部署等关键环节。

二、Ubuntu 服务器建设

（一）服务器选择

在搭建服务器时，有两种可行方案：

1. **使用三方云服务器**：这种方式需要支付一定费用，但能获得稳定的网络环境和强大的性能支撑，适合对服务器性能和稳定性要求较高的项目。
2. **使用个人笔记本**：手动安装 Linux 操作系统（如 CentOS、Ubuntu 等），此方法可节省成本，但可能受限于个人笔记本的硬件性能，适用于测试或小型项目。

（二）服务器开启 SSH 连接

SSH 连接是远程管理服务器的重要手段，以下是开启 SSH 连接的详细步骤：

```
# 检查SSH当前服务状态
sudo systemctl status ssh
# 检查是否安装SSH服务
dpkg -l | grep openssh-server
# 更新软件源
sudo apt-get update
# 下载安装ssh服务
sudo apt-get install openssh-server
# 启动ssh服务
sudo systemctl start ssh
# 设置ssh服务开机自启动
sudo systemctl enable ssh
# 查看默认端口（默认port=22）
```

```
sudo cat /etc/ssh/ssh_config
# 确保你的 Ubuntu 服务器的防火墙允许 SSH 连接（默认端口是 22）
sudo ufw allow ssh
sudo ufw enable
```

（三）使用三方工具（如 iSH Pro）远程连接

以下是在 iSH Pro 上设置和连接 Ubuntu 服务器的详细步骤：

1. **打开 iSH Pro 应用**：在设备主屏幕找到并打开 iSH Pro 应用程序。
2. **创建新的 SSH 会话**：在 iSH Pro 中，通常会看到一个“+”按钮，点击它并选择“New Session”。
3. 配置 SSH 会话
   ：在新建会话的配置界面，需要输入以下信息：
   - **Session Name**：输入一个容易识别的名称，例如“Ubuntu Server”。
   - **Hostname**：输入你的 Ubuntu 服务器的 IP 地址。
   - **Username**：输入用于登录 Ubuntu 服务器的用户名。
   - **Password**：输入密码（如果使用密码认证）；若选择使用 SSH 密钥进行认证，则需要将公钥添加到服务器的 `~/.ssh/authorized_keys` 文件中。
4. **保存并连接**：配置完成后，保存会话设置，然后点击连接图标开始连接。
5. **首次连接处理**：首次连接时，iSH Pro 会提示接受服务器的公钥指纹，确认无误后接受即可。

使用 SSH 密钥（更安全）

为了提高连接的安全性，建议使用 SSH 密钥进行认证，步骤如下：

1. **在 iOS 设备上生成 SSH 密钥对**（如果还没有）：

```
ssh-keygen -t rsa -b 2048
```

按照提示操作，可将密钥保存到默认位置或指定位置。

2. **将公钥添加到 Ubuntu 服务器的 `~/.ssh/authorized_keys` 文件中**：

shell

```
cat ~/.ssh/id_rsa.pub | ssh user@your_server_ip 'cat >>
.ssh/authorized_keys'
```

请将 `user` 和 `your_server_ip` 替换为你的用户名和服务器 IP。

3. 在 **iSH Pro 中配置 SSH 会话**：选择使用密钥认证，并指向你的私钥文件。

通过以上步骤，你应该能够在 iSH Pro 上成功连接到你的 Ubuntu 服务器。

三、MySQL 数据库安装与使用

（一）数据库安装

以下是安装 MySQL 服务包的步骤：

```
# 更新软件源
sudo apt-get update
# 下载安装mysql服务包
sudo apt-get install mysql-server
# 启动mysql服务并设置开机启动
sudo systemctl start mysql
sudo systemctl enable mysql
# 找到mysql的登录密码（可以通过日志文件登录），默认为用户登录密码
sudo cat /var/log/mysql/error.log | grep 'temporary password'
```

（二）数据库权限配置

```
-- 设置新密码
ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql_native_password
BY '新密码';
-- 刷新权限
FLUSH PRIVILEGES;

-- 创建数据库(项目使用的db name)
CREATE DATABASE mydata CHARACTER SET utf8mb4 COLLATE
utf8mb4_unicode_ci;

-- 创建用户可以从任何主机连接数据库
CREATE USER 'root'@'%' IDENTIFIED BY '123456';
-- 授权用户访问数据库
GRANT ALL PRIVILEGES ON mydata.* TO 'root'@'%';
-- 刷新权限
FLUSH PRIVILEGES;

-- 如果远程连接不上，可通过修改认证插件
-- 修改 root@localhost 的认证插件为 mysql_native_password
ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql_native_password
BY '123456';
ALTER USER 'root'@'%' IDENTIFIED WITH mysql_native_password BY
'123456';

FLUSH PRIVILEGES;
```

四、本地开发代码上传码云

（一）初始化本地代码文件夹

（二）添加.gitignore 文件

添加 `.gitignore` 文件可以告诉 Git 在 `add` 和 `push` 操作时忽略某些文件和文件夹。以下是一个示例：

```
# 忽略虚拟环境
.venv/
venv/

# 忽略 SQLite 数据库文件
db.sqlite3

# 忽略构建目录
build/

# 忽略日志文件
*.log

# 忽略 Python 字节码
__pycache__/
*.pyc

# 忽略系统文件
.DS_Store
Thumbs.db

# 忽略 git 相关的临时文件
*.git/
```

（三）多人协同操作

多人协同开发的详细信息请参考以下链接：
https://blog.csdn.net/yeye_queenmoon/article/details/144472289

五、远程服务器部署

（一）设置秘钥并拉取项目

后续通过 SSH 拉取项目，使用以下命令：

1. **确认 SSH 密钥已配置**
   确保服务器已配置 GitHub 的 SSH 密钥（避免每次拉取时输入密码）：

```
# 检查是否存在SSH密钥
ls ~/.ssh/id_rsa.pub
# 若不存在，生成新密钥（按提示操作）
ssh-keygen -t rsa -b 4096 -C "your_email@example.com"
```

   将公钥（`~/.ssh/id_rsa.pub`）添加到 GitHub 账户的SSH 密钥设置中。

2. **测试 SSH 连接**

```
ssh -T git@github.com
# 若返回 "Welcome to GitHub..."，说明配置成功
```

拉取项目环境

```
git clone https://github.com/yourusername/yourproject.git
```

（二）配置所需环境

1. 安装依赖项

为确保项目能够正常构建，需要安装一些依赖项：

```
# 更新链接库
sudo apt-get update
# 安装venv环境库
sudo apt-get install python3-venv python3-dev curl
# sqlclient的依赖库
sudo apt-get install libmysqlclient-dev
# 安装其他库包括GCC等
sudo apt install -y build-essential libssl-dev zlib1g-dev
libncurses5-dev libgdbm-dev libnss3-dev libsqlite3-dev libreadline-
dev libffi-dev libbz2-dev
```

2. 创建 venv 环境

进入项目根目录并创建虚拟环境：

```
cd yourproject  # 进入项目根目录
python3 -m venv venv
source venv/bin/activate

# 激活虚拟环境后，安装新的 `pip` 以及项目所需的依赖（根据提示安装不同的依赖环境）
pip install -r requirements.txt
```

3. 使用 Gunicorn + Nginx（生产环境）

安装 Gunicorn

```
pip install gunicorn
```

配置 Gunicorn

```
# 创建gunicorn.service
sudo vim /etc/systemd/system/gunicorn.service

# gunicorn.service写入以下内容
######################
[Unit]
Description=gunicorn daemon
Requires=gunicorn.socket
After=network.target
```

```
[Service]
; 主机用户名
User=www-data
Group=www-data
WorkingDirectory=/home/shin/device-manager-system   ; 工作目录
ExecStart=/home/shin/device-manager-system/venv/bin/gunicorn \  ;
VENEV环境
          --access-logfile - \
          --workers 3 \
          --bind unix:/run/gunicorn.sock \
          device_manager.wsgi:application

[Install]
WantedBy=multi-user.target
######################
# 保存并退出文件

# 创建gunicorn.socket套接字文件
sudo vim /etc/systemd/system/gunicorn.socket

# 套接字文件写入以下内容
######################
[Unit]
Description=gunicorn socket

[Socket]
ListenStream=/run/gunicorn.sock

[Install]
WantedBy=sockets.target
######################
# 保存并退出文件
```

启动Gunicorn服务

```
# 确保权限问题
sudo chown -R www-data:www-data /home/shin/device-manager-system
sudo chmod -R 755 /home/shin/device-manager-system

# 启动并启用 Gunicorn 服务和套接字
sudo systemctl start gunicorn.socket
sudo systemctl enable gunicorn.socket

# 查看Gunicorn日志(debug使用)
sudo journalctl -u gunicorn -f
```

安装 Nginx

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

配置Nginx

```
# 创建 Nginx 配置文件
sudo vim /etc/nginx/sites-available/yourproject

# 写入以下内容
#####################
server {
    listen 80;
    server_name 192.168.0.214;  # 使用实际的服务器IP

    # 静态文件配置（已修正路径）
    location /static/ {
        alias /home/shin/device-manager-system/collected_static/;
 # 与STATIC_ROOT一致
        expires 30d;
        access_log off;
        add_header Cache-Control "public";
    }

    # 媒体文件配置
    location /media/ {
        alias /home/shin/device-manager-system/media/;
        expires 30d;
```

```
        access_log off;
        add_header Cache-Control "public";
    }

    # Gunicorn反向代理配置（添加超时和错误处理）
    location / {
        include proxy_params;
        proxy_pass http://unix:/run/gunicorn.sock;

        # 添加超时设置避免504错误
        proxy_connect_timeout 60s;
        proxy_send_timeout 120s;
        proxy_read_timeout 120s;

        # 错误页面处理
        error_page 500 502 503 504 /500.html;
        location = /500.html {
            root /var/www/html;
        }
    }
}
######################
# 保存并退出文件

# 创建软链接
sudo ln -s /etc/nginx/sites-available/yourproject /etc/nginx/sites-
enabled
```

启动Nginx

```
# 测试 Nginx 配置
sudo nginx -t

# 如果配置无误，重启 Nginx 服务
sudo systemctl restart nginx

# 查看 Nginx 错误日志（debug使用）
sudo tail -f /var/log/nginx/error.log
```

六、项目运行以及Debug

通过以上步骤，你可以成功在 Ubuntu 服务器上部署 Django 项目。

在浏览器中访问 `http://<远程服务器 IP 地址>` 即可查看项目。

检查项

1.检查Nginx和UWsgi服务状态

```
# 检查Nginx和Gunicorn运行状态
sudo systemctl status gunicorn

# 检查Nginx和Gunicorn运行状态
sudo systemctl status nginx
```

2.检查项目文件执行权限

为确保项目正常运行，需要设置项目目录权限并确保虚拟环境可执行：

```
# 项目目录权限
sudo chown -R www-data:www-data /home/shin/device-manager-system/
chmod 755 /home/shin/device-manager-system

# 静态文件目录（Nginx可直接访问）
chmod -R 755 /home/shin/device-manager-system/static

# 媒体文件目录（可写）
chmod -R 775 /home/shin/device-manager-system/media
chown -R www-data:www-data /home/shin/device-manager-system/media

### 此处后续有待修改
```

3.检查服务器防火墙允许

为了允许 HTTP 和 HTTPS 流量，需要修正防火墙规则：

```
# 修正防火墙规则
sudo ufw allow 80  # Nginx
sudo ufw allow 443 # Gun
sudo ufw reload
```

七、创建定时代码更新

（一）、准备工作

1. **确认 SSH 密钥已配置**
   确保服务器已配置 GitHub 的 SSH 密钥（避免每次拉取时输入密码）：

```
# 检查是否存在SSH密钥
ls ~/.ssh/id_rsa.pub
# 若不存在，生成新密钥（按提示操作）
ssh-keygen -t rsa -b 4096 -C "your_email@example.com"
```

   将公钥（`~/.ssh/id_rsa.pub`）添加到 GitHub 账户的SSH 密钥设置中。

2. **测试 SSH 连接**

```
ssh -T git@github.com
# 若返回 "Welcome to GitHub..."，说明配置成功
```

（二）、创建定时拉取脚本

为避免命令复杂，建议创建脚本文件执行拉取操作：

```
# 创建脚本文件
sudo vim /home/shin/device_manager_system/pull_device-manager-
system.sh
```

写入以下内容（根据实际项目修改路径）：

```
#!/bin/bash

PROJECT_DIR="/home/shin/device-manager-system"
LOG_FILE="/home/shin/tasks/pull_device-manager-system_log.txt"

echo "=== $(date) ===" >> "$LOG_FILE"

# 检查项目目录
cd "$PROJECT_DIR" || { echo "错误：无法进入项目目录！" >> "$LOG_FILE";
exit 1; }

# 检查虚拟环境
```

```
if [ ! -d "$PROJECT_DIR/venv" ]; then
    echo "错误：虚拟环境不存在！" >> "$LOG_FILE"
    exit 1
fi

# 拉取代码前记录提交ID
OLD_COMMIT=$(git rev-parse HEAD)

# 拉取最新代码
git pull origin main >> "$LOG_FILE" 2>&1
if [ $? -ne 0 ]; then
    echo "错误：拉取代码失败！" >> "$LOG_FILE"
    exit 1
fi

# 拉取后记录新提交ID
NEW_COMMIT=$(git rev-parse HEAD)

# 检查是否有新提交
if [ "$OLD_COMMIT" = "$NEW_COMMIT" ]; then
    echo "没有新代码更新，跳过后续部署步骤" >> "$LOG_FILE"
    exit 0
fi

echo "检测到代码更新，开始部署..." >> "$LOG_FILE"

# 安装依赖（无密码sudo需提前配置）
echo "正在更新依赖..." >> "$LOG_FILE"
sudo -u www-data "$PROJECT_DIR/venv/bin/pip" install --upgrade -r
requirements.txt >> "$LOG_FILE" 2>&1
if [ $? -ne 0 ]; then
    echo "错误：安装依赖失败！" >> "$LOG_FILE"
    exit 1
fi

# 收集静态文件（无密码sudo需提前配置）
echo "正在收集静态文件..." >> "$LOG_FILE"
sudo -u www-data "$PROJECT_DIR/venv/bin/python"
"$PROJECT_DIR/manage.py" collectstatic --noinput >> "$LOG_FILE"
2>&1
if [ $? -ne 0 ]; then
    echo "错误：收集静态文件失败！" >> "$LOG_FILE"
```

```
    exit 1
fi

# 重启gunicorn服务
echo "正在重启gunicorn服务..." >> "$LOG_FILE"
sudo systemctl restart gunicorn >> "$LOG_FILE" 2>&1
if [ $? -ne 0 ]; then
    echo "错误：重启gunicorn失败！" >> "$LOG_FILE"
    exit 1
fi

# 重启Nginx服务
echo "正在重启Nginx服务..." >> "$LOG_FILE"
sudo systemctl restart nginx >> "$LOG_FILE" 2>&1
if [ $? -ne 0 ]; then
    echo "错误：重启Nginx失败！" >> "$LOG_FILE"
    exit 1
fi

echo "部署成功！" >> "$LOG_FILE"
exit 0
```

保存后赋予脚本执行权限：

```
# uWSGI 通常以 www-data 用户运行，而你可能以本地用户执行 Git 操作，导致文件所有权冲突
# 允许 shin 用户以 www-data 身份无密码运行命令
# 打开终端编辑sudoers文件
sudo visudo
# 写入以下命令（建议使用最小权命令）
shin ALL=(www-data) NOPASSWD: ALL # 允许shin以www-data无密码执行所有命令
shin ALL=(www-data) NOPASSWD: /home/shin/device-manager-system/venv/bin/pip, /home/shin/device-manager-system/venv/bin/python # 仅针对需要的执行的命令进行赋权，避免all的出现
shin ALL=(root) NOPASSWD: /bin/systemctl restart gunicorn, /bin/systemctl restart nginx # 允许shin用户以root用户执行特定命令

###
#解释：
# shin：当前执行脚本的用户。
# (root)/(www-data)：以 root 身份运行命令。
```

```
# NOPASSWD:：免密码执行。
# /bin/systemctl restart gunicorn：允许无密码运行的命令路径。
###



# 为当前用户（shin）添加读写权限
sudo setfacl -R -m u:shin:rwX /home/shin/device-manager-system

# 定时脚本服务执行权限
chmod +x /home/shin/device_manager_system/pull_device-manager-
system.sh



### 此处后续有待修改
```

（三）、配置 crontab 定时任务

1. 编辑 **crontab**

```
crontab -e
```

2. 添加定时任务

在文件末尾添加以下行（示例为每天凌晨 2 点拉取）：

```
# 每天2:00拉取GitHub代码
0 2 * * * /home/shin/tasks/pull_device-manager-system.sh
```

定时规则说明（从左到右）：

- 分钟（0-59）| 小时（0-23）| 日期（1-31）| 月份（1-12）| 星期（0-6，0 为周日）

  常用示例：

- 每 10 分钟拉取：`*/10 * * * * /home/shin/pull_device-manager-system.sh ...`
- 每周一 8 点拉取：`0 8 * * 1 /home/shin/pull_device-manager-system.sh ...`

（四）、验证定时任务

1. 检查 **crontab** 条目

```
crontab -l
# 确认已添加的定时任务
```

2. 手动执行脚本测试

```
/home/shin/pull_github_project.sh
# 查看日志文件是否有输出
cat /home/shin/pull_log.txt
```

3. 模拟定时任务触发

```
# 查看当前时间
date
# 修改系统时间测试（测试后恢复时间）
sudo date -s "2025-06-17 02:00:00"
# 等待片刻后查看日志
tail -f /home/shin/pull_log.txt
```