Python

- Wagtail 部署指南
- GitHub 高赞 Python3 Web 项目推荐
- Pyenv 使用完整指南
- Python 使用 MySQL 数据库完整指南

Wagtail部署指南

Wagtail是一个基于Django的现代化CMS系统，以下是部署Wagtail的详细步骤，涵盖多种部署方式。

1. 本地开发环境部署

前置要求

- Python 3.7+
- PostgreSQL/MySQL/SQLite (推荐PostgreSQL)
- Node.js (如需前端构建)

安装步骤

```
# 创建虚拟环境
python -m venv venv
source venv/bin/activate  # Linux/Mac
# venv\Scripts\activate   # Windows

# 安装 Wagtail
pip install wagtail

# 创建新项目
wagtail start mysite
cd mysite

# 安装依赖
pip install -r requirements.txt

# 初始化数据库
python manage.py migrate

# 创建超级用户
python manage.py createsuperuser
```

```
# 运行开发服务器
python manage.py runserver
```

访问 http://localhost:8000 查看站点， http://localhost:8000/admin 访问后台。

2. 生产环境部署 (Docker 方式)

docker-compose.yml 示例

```
version: '3.8'

services:
  db:
    image: postgres:13
    environment:
      POSTGRES_DB: wagtail
      POSTGRES_USER: wagtail
      POSTGRES_PASSWORD: wagtail
    volumes:
      - postgres_data:/var/lib/postgresql/data

  web:
    build: .
    command: gunicorn mysite.wsgi:application --bind 0.0.0.0:8000
    volumes:
      - .:/code
    ports:
      - "8000:8000"
    depends_on:
      - db
    environment:
      DATABASE_URL: postgres://wagtail:wagtail@db:5432/wagtail
      SECRET_KEY: your-secret-key-here

volumes:
  postgres_data:
```

Dockerfile 示例

```
FROM python:3.9-slim

WORKDIR /code

ENV PYTHONDONTWRITEBYTECODE 1
ENV PYTHONUNBUFFERED 1

RUN apt-get update && apt-get install -y \
    libpq-dev \
    gcc \
    && rm -rf /var/lib/apt/lists/*

COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

COPY . .

EXPOSE 8000
```

部署步骤

```
# 构建并启动容器
docker-compose up -d --build

# 执行数据库迁移
docker-compose exec web python manage.py migrate

# 创建超级用户
docker-compose exec web python manage.py createsuperuser

# 收集静态文件
docker-compose exec web python manage.py collectstatic --no-input
```

3. 生产环境部署 (传统服务器方式)

□□ Gunicorn + Nginx

1. □□ Gunicorn:

```
pip install gunicorn
```

2. □□ Gunicorn □□□□ `/etc/systemd/system/gunicorn.service`:

```
[Unit]
Description=gunicorn daemon
After=network.target

[Service]
User=youruser
Group=www-data
WorkingDirectory=/path/to/your/project
ExecStart=/path/to/venv/bin/gunicorn --access-logfile - --workers 3 --bind unix:/run/gunicorn.sock
mysite.wsgi:application

[Install]
WantedBy=multi-user.target
```

3. Nginx □□□□ (`/etc/nginx/sites-available/yourdomain`):

```
server {
    listen 80;
    server_name yourdomain.com;

    location = /favicon.ico { access_log off; log_not_found off; }

    location /static/ {
        root /path/to/your/project;
    }

    location /media/ {
        root /path/to/your/project;
    }

    location / {
        include proxy_params;
```

```
        proxy_pass http://unix:/run/gunicorn.sock;
    }
}
```

4. 启用配置：

```
sudo ln -s /etc/nginx/sites-available/yourdomain /etc/nginx/sites-enabled
sudo systemctl restart nginx
```

4. 云平台部署

Heroku 部署

1. 创建 `Procfile`：

```
web: gunicorn mysite.wsgi:application --bind 0.0.0.0:$PORT
```

2. 创建 `runtime.txt` 指定 Python 版本：

```
python-3.9.7
```

3. 部署命令：

```
heroku create
heroku addons:create heroku-postgresql:hobby-dev
heroku config:set SECRET_KEY=your-secret-key
heroku config:set DISABLE_COLLECTSTATIC=1
git push heroku main
heroku run python manage.py migrate
heroku run python manage.py createsuperuser
```

AWS Elastic Beanstalk 部署

1. 创建 `requirements.txt` 包含所有依赖
2. 创建 `.ebextensions/django.config`：

```
option_settings:
  aws:elasticbeanstalk:container:python:
```

```
    WSGIPath: mysite/wsgi.py
```

3. 部署命令：

```
eb init -p python-3.9 wagtail-app
eb create wagtail-env
```

5. 部署后配置

1. 安全配置：

```
# settings.py
DEBUG = False
ALLOWED_HOSTS = ['yourdomain.com', 'localhost']
SECURE_SSL_REDIRECT = True
SESSION_COOKIE_SECURE = True
CSRF_COOKIE_SECURE = True
```

2. 性能优化：
   - 设置缓存
   - 启用数据库连接池
   - 配置静态文件CDN
3. 备份策略：

```
# 数据库备份
docker-compose exec db pg_dump -U wagtail wagtail > backup.sql

# 媒体文件备份
tar -czvf media_backup.tar.gz /path/to/media
```

6. 常见问题解决

1. 静态文件**404**错误：
   - 确保执行了 `collectstatic`
   - 检查Nginx/Apache配置中的静态文件路径
2. 数据库连接问题：
   - 检查数据库服务是否运行
   - 验证 `DATABASE_URL` 或 `settings.py` 中的配置
3. 性能问题：

- 增加 Gunicorn worker 数量（通常 2-4 * CPU核心）
- 添加缓存层 (Redis/Memcached)

通过以上步骤，您应该能够成功部署 Wagtail CMS 到各种环境中。根据您的具体需求选择最适合的部署方式。

GitHub高赞Python3 Web项目推荐

以下是GitHub上star数量较高的Python3 Web项目，涵盖了各种框架和实用场景：

1. 综合类Web项目

awesome-python (□ 180k+)

- 不是具体项目，而是Python资源的终极精选列表
- 包含Web框架、爬虫、数据分析等所有领域的优秀项目

public-apis (□ 275k+)

- 免费API集合项目
- 使用Python构建的Web API示例参考

2. Django项目

django-allauth (□ 8.5k+)

- 集成认证系统，支持多种社交账号登录
- 学习Django认证系统的优秀案例

saleor (□ 19k+)

- 基于Django和GraphQL的电商平台

- 现代电商系统架构参考

wagtail (□ 16k+)

- Django 内容管理系统 (CMS)
- 企业级 CMS 实现范例

3. Flask 项目

flasky (□ 8k+)

- 《Flask Web开发》一书配套代码
- Flask 学习最佳实践

cookiecutter-flask (□ 4.5k+)

- Flask 项目模板生成器
- 规范化 Flask 项目结构参考

4. FastAPI 项目

fastapi (□ 67k+)

- FastAPI 官方仓库
- 现代 Python Web API 开发典范

full-stack-fastapi-postgresql (□ 14k+)

- 全栈 FastAPI + PostgreSQL 项目模板
- 包含前端 (Vue)、后端和部署配置

5. 其他有趣项目

langchain (□ 74k+)

- 使用Python构建LLM应用的框架
- 包含Web接口和API设计

streamlit (□ 30k+)

- 快速创建数据科学Web应用
- 将Python脚本转换为可分享的Web应用

spaCy (□ 28k+)

- 工业级自然语言处理库
- 包含Web服务和API示例

学习建议

1. 从简单项目开始：先学习flasky或fastapi官方示例
2. 研究项目结构：查看cookiecutter生成的项目模板
3. 深入业务逻辑：分析saleor这样的完整商业项目
4. 关注技术组合：观察这些项目如何集成数据库、缓存、消息队列等
5. 参与贡献：从文档改进或简单bug修复开始参与开源项目

这些高星项目代表了Python Web开发的最佳实践，通过阅读它们的代码和文档可以快速提升开发能力。

Pyenv 使用完整指南

Pyenv 是一个强大的 Python 版本管理工具，可以让你轻松安装、切换和管理多个 Python 版本。以下是详细使用指南：

1. 安装 Pyenv

Linux/macOS 安装

```
curl https://pyenv.run | bash
```

安装完成后，将以下内容添加到 shell 配置文件（如 `~/.bashrc`，`~/.zshrc` 或 `~/.bash_profile`）：

```
export PYENV_ROOT="$HOME/.pyenv"
export PATH="$PYENV_ROOT/bin:$PATH"
eval "$(pyenv init -)"
eval "$(pyenv virtualenv-init -)"  # 如果要用虚拟环境插件
```

然后重新加载 shell：

```
exec "$SHELL"
```

Windows 安装

Windows 用户可以使用 pyenv-win：

```
Invoke-WebRequest -UseBasicParsing -Uri "https://raw.githubusercontent.com/pyenv-win/pyenv-win/master/pyenv-win/install-pyenv-win.ps1" -OutFile "./install-pyenv-win.ps1"; &"./install-pyenv-win.ps1"
```

2. 基本命令

安装 Python 版本

```
# 查看可安装版本
pyenv install --list

# 安装特定版本
pyenv install 3.9.7

# 安装最新稳定版
pyenv install $(pyenv latest -k 3)
```

查看已安装版本

```
pyenv versions
```

带 `*` 的表示当前激活的版本。

切换 Python 版本

```
# 全局设置（影响整个系统）
pyenv global 3.9.7

# 局部设置（仅当前目录及其子目录）
pyenv local 3.8.12

# shell会话级别设置（仅当前终端）
pyenv shell 3.10.0
```

卸载 Python 版本

```
pyenv uninstall 3.7.12
```

3. 高级功能

虚拟环境管理

Pyenv 可以与 `pyenv-virtualenv` 插件配合使用:

```
# 创建虚拟环境
pyenv virtualenv 3.9.7 my-project-env


# 激活虚拟环境
pyenv activate my-project-env


# 停用虚拟环境
pyenv deactivate


# 删除虚拟环境
pyenv uninstall my-project-env
```

版本优先级

Pyenv 按以下顺序决定使用哪个 Python 版本:

1. `PYENV_VERSION` 环境变量
2. 当前目录的 `.python-version` 文件 (由 `pyenv local` 创建)
3. 第一个找到的父目录中的 `.python-version` 文件
4. 全局版本 (由 `pyenv global` 设置)

自定义构建选项

```
# 安装时启用优化
PYTHON_CONFIGURE_OPTS="--enable-optimizations" pyenv install 3.9.7


# 安装特定架构版本
env PYTHON_CONFIGURE_OPTS="--enable-shared" pyenv install 3.10.0
```

4. 常见问题解决

安装失败 (缺少依赖)

```
# Ubuntu/Debian
sudo apt-get install -y make build-essential libssl-dev zlib1g-dev \
libbz2-dev libreadline-dev libsqlite3-dev wget curl llvm \
libncursesw5-dev xz-utils tk-dev libxml2-dev libxmlsec1-dev libffi-dev liblzma-dev

# CentOS/RHEL
sudo yum install -y gcc zlib-devel bzip2 bzip2-devel readline-devel \
sqlite sqlite-devel openssl-devel tk-devel libffi-devel
```

速度慢的解决方法

使用国内镜像:

```
# 临时使用镜像
v=3.9.7;wget https://npm.taobao.org/mirrors/python/$v/Python-$v.tar.xz -P ~/.pyenv/cache/;pyenv install $v

# 或设置环境变量
export PYTHON_BUILD_MIRROR_URL="https://npm.taobao.org/mirrors/python"
```

更新 Pyenv

```
pyenv update
```

5. 实际应用示例

项目中使用特定 Python 版本

```
# 进入项目目录
cd my-project

# 设置项目使用的 Python版本
pyenv local 3.8.12

# 创建项目专用虚拟环境
pyenv virtualenv 3.8.12 my-project-env
```

```
# 激活环境
pyenv activate my-project-env

# 安装依赖
pip install -r requirements.txt
```

多版本切换演示

```
# 查看当前版本
python --version

# 切换到另一个版本
pyenv shell 3.7.12
python --version

# 切换回来
pyenv shell --unset
python --version
```

6. 配置建议

在 `~/.pyenv/` 目录下可以配置一些默认行为:

- `plugins/python-build/share/python-build/` - 可以添加自定义的 Python 版本定义
- `cache/` - 缓存下载的 Python 源码包

7. 与其他工具集成

与 Pipenv 配合使用

```
pyenv local 3.9.7
pip install --user pipenv
pipenv install
```

与 Poetry 配合使用

```
pyenv local 3.10.0
curl -sSL https://install.python-poetry.org | python -
poetry init
```

Pyenv 提供了灵活的 Python 版本管理，特别适合需要同时处理多个项目的开发者。

Python使用MySQL数据库完整指南

Python提供了多种方式连接和操作MySQL数据库，以下是常用的方法和最佳实践。

1. 安装MySQL驱动

常用驱动选择

```
# 最流行的纯Python驱动
pip install mysql-connector-python

# 更快的C扩展驱动（推荐）
pip install pymysql

# MySQL官方驱动
pip install mysql-connector-python

# 另一个流行选择
pip install MySQL-python  # 注意：仅支持Python 2.x和旧版3.x
```

2. 连接MySQL数据库

使用PyMySQL(推荐)

```
import pymysql

# 建立连接
connection = pymysql.connect(
```

```
    host='localhost',
    user='username',
    password='password',
    database='dbname',
    charset='utf8mb4',
    cursorclass=pymysql.cursors.DictCursor  # 返回字典格式结果
)

try:
    with connection.cursor() as cursor:
        # 执行  SQL查询
        sql = "SELECT * FROM `users` WHERE `email`=%s"
        cursor.execute(sql, ('user@example.com',))

        # 获取结果
        result = cursor.fetchone()
        print(result)
finally:
    connection.close()
```

使用 mysql-connector-python（官方驱动 ）

```
import mysql.connector

config = {
    'user': 'username',
    'password': 'password',
    'host': 'localhost',
    'database': 'dbname',
    'raise_on_warnings': True
}

cnx = mysql.connector.connect(**config)
cursor = cnx.cursor(dictionary=True)  # 返回字典格式结果

query = "SELECT * FROM employees WHERE hire_date > %s"
hire_start = datetime.date(1999, 1, 1)
cursor.execute(query, (hire_start,))
```

```
for row in cursor:
    print(row)

cursor.close()
cnx.close()
```

3. CRUD □□□

□□□

```
def create_table():
    conn = pymysql.connect(host='localhost', user='root', password='', database='test')
    try:
        with conn.cursor() as cursor:
            sql = """
            CREATE TABLE IF NOT EXISTS `users` (
                `id` INT AUTO_INCREMENT PRIMARY KEY,
                `name` VARCHAR(255) NOT NULL,
                `email` VARCHAR(255) NOT NULL UNIQUE,
                `created_at` TIMESTAMP DEFAULT CURRENT_TIMESTAMP
            ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
            """
            cursor.execute(sql)
        conn.commit()
    finally:
        conn.close()
```

□□□

```
def insert_user(name, email):
    conn = pymysql.connect(host='localhost', user='root', password='', database='test')
    try:
        with conn.cursor() as cursor:
            sql = "INSERT INTO `users` (`name`, `email`) VALUES (%s, %s)"
```

```
            cursor.execute(sql, (name, email))
        conn.commit()
        return cursor.lastrowid
    except pymysql.err.IntegrityError:
        print("Email already exists")
        return None
    finally:
        conn.close()
```

```
def get_users(page=1, per_page=10):
    conn = pymysql.connect(host='localhost', user='root', password='', database='test')
    try:
        with conn.cursor() as cursor:
            offset = (page - 1) * per_page
            sql = "SELECT * FROM `users` LIMIT %s OFFSET %s"
            cursor.execute(sql, (per_page, offset))
            return cursor.fetchall()
    finally:
        conn.close()
```

```
def update_user(user_id, name=None, email=None):
    conn = pymysql.connect(host='localhost', user='root', password='', database='test')
    try:
        with conn.cursor() as cursor:
            updates = []
            params = []
            if name:
                updates.append("`name` = %s")
                params.append(name)
            if email:
                updates.append("`email` = %s")
                params.append(email)
```

```
            if updates:
                params.append(user_id)
                sql = f"UPDATE `users` SET {', '.join(updates)} WHERE `id` = %s"
                cursor.execute(sql, params)
                conn.commit()
                return cursor.rowcount
            return 0
    except pymysql.err.IntegrityError:
        print("Email already exists")
        return None
    finally:
        conn.close()
```

□□□

```
def delete_user(user_id):
    conn = pymysql.connect(host='localhost', user='root', password='', database='test')
    try:
        with conn.cursor() as cursor:
            sql = "DELETE FROM `users` WHERE `id` = %s"
            cursor.execute(sql, (user_id,))
            conn.commit()
            return cursor.rowcount
    finally:
        conn.close()
```

4. □□ ORM (SQLAlchemy)

```
from sqlalchemy import create_engine, Column, Integer, String, DateTime
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker
from datetime import datetime

# □□□□□□□ : mysql+pymysql://<username>:<password>@<host>/<dbname>[?<options>]
engine = create_engine('mysql+pymysql://root:@localhost/test?charset=utf8mb4')
Base = declarative_base()
```

```
class User(Base):
    __tablename__ = 'users'

    id = Column(Integer, primary_key=True)
    name = Column(String(255), nullable=False)
    email = Column(String(255), unique=True, nullable=False)
    created_at = Column(DateTime, default=datetime.now)

# 创建表
Base.metadata.create_all(engine)

# 创建会话
Session = sessionmaker(bind=engine)
session = Session()

# 添加新用户
new_user = User(name="John Doe", email="john@example.com")
session.add(new_user)
session.commit()

# 查询用户
user = session.query(User).filter_by(email="john@example.com").first()
print(user.name, user.email)

session.close()
```

5. 连接池管理

使用 DBUtils 创建连接池

```
from dbutils.pooled_db import PooledDB
import pymysql

pool = PooledDB(
    creator=pymysql,
```

```
    maxconnections=10,
    mincached=2,
    host='localhost',
    user='root',
    password='',
    database='test',
    charset='utf8mb4'
)

def get_users():
    conn = pool.connection()
    try:
        with conn.cursor() as cursor:
            cursor.execute("SELECT * FROM users")
            return cursor.fetchall()
    finally:
        conn.close()
```

6. 最佳实践

1. 始终使用参数化查询 - 防止 SQL 注入

```
# 错误做法
cursor.execute(f"SELECT * FROM users WHERE name = '{name}'")

# 正确做法
cursor.execute("SELECT * FROM users WHERE name = %s", (name,))
```

2. 使用上下文管理器 - 自动关闭连接

```
with pymysql.connect(...) as conn:
    with conn.cursor() as cursor:
        cursor.execute(...)
```

3. 处理事务 - 确保数据一致性

```
try:
    conn.begin()
    # 执行多个操作
```

```
    conn.commit()
except:
    conn.rollback()
    raise
```

4. 设置合适的字符集 - 推荐使用 `utf8mb4` 支持完整 Unicode
5. 连接管理 - 生产环境使用连接池
6. 错误处理 - 捕获特定异常

```
try:
    cursor.execute(...)
except pymysql.err.IntegrityError as e:
    print("Duplicate entry:", e)
except pymysql.err.ProgrammingError as e:
    print("SQL syntax error:", e)
```

7. 性能优化

1. 使用索引 - 确保查询字段有适当索引
2. 批量操作 - 减少数据库往返

```
# 批量插入
sql = "INSERT INTO users (name, email) VALUES (%s, %s)"
cursor.executemany(sql, [('a','a@a.com'), ('b','b@b.com')])
```

3. 只查询需要的列 - 避免 `SELECT *`
4. 使用连接池 - 减少连接创建开销
5. 合理设置缓存 - 对不常变的数据使用缓存

通过以上方法，你可以在 Python 中高效安全地使用 MySQL 数据库。