部署 Flask 应用

原创 太阳雪 Python技术 3月9日

Web 应用只有部署到服务器上才能被真正的使用，前面我们了解了用 Flask 开发 Web 应用，今天就来了解下，如何部署 Flask 应用。

与开发应用相比，部署应用，多些工作，如处理日志，服务器状态报告，项目打包等，接下来我们逐个了解下。

日志简介

在练习 Web 应用时，对日志的需求不高，程序问题可以在调试时看到。对于发布后的应用，没法实时查看运行信息，为了方便查错，需要将运行信息记录在日志里。

logging 模块是 Python 中 常用的日志处理模块，可以支持多种日志输出形式，如文本、邮件、日志服务器等，日志被分为 5 个等级，用于不同的情况，等级从低到高依次是：

- `debug`：用于开发时调试
- `info`：用于记录必要信息，以便了解程序运行状态
- `warning`：警告，适用于轻微的不影响系统运行的错误提示
- `error`：错误，用于不能忽略的异常情况，计算错误，处理流程出错等
- `critical`：严重错误，能会使应用挂掉或者不可恢复的严重问题

调用与等级名称相同的方法，就可以记录相应等级的日志，例如：

```
logger.debug('我是一个调试信息')
logger.warning('我是一个警告')
```

等级除了用于日志分类外，还可以作为输出的过滤条件，例如只记录 warning 级以上的日志：

```
logger.setLevel(level = logging.WARNING)
```

另外，还可以定义出入内容和样式，例如：

```
logging.basicConfig(format = '%(asctime)s - %(name)s - %(levelname)s - %(message)s')
```

Flask 应用实例，带有日志实例 logger，使用很方便，例如：

```python
from flask import Flask

app = Flask(__name__)

app.logger.info("Hello logger!")  # 记录一个等级为 info 的日志
```

在 blueprint 中，可以通过引入 current_app，即当前环境中的应用实例来得到 logger，如：

```python
from flask import Blueprint
from flask import current_app

home_bp = Blueprint('home_bp', __name__)
@home_bp.route('/', methods=['GET', 'POST'])
def index():
    current_app.logger.debug("进入首页视图函数处理")
```

日志配置

应用需要部署在不同的环境中，一般分开发环境，测试环境，生产环境，不同的环境对日志记录的需求不同，例如开发环境，需要详细的日志，生产环境需要简洁的日志，且能将严重错误通知管理员等等。

在 《Flask 项目结构》一节中，我们使用 config 类为不同环境设置不同数据库连接，也可以为不同环境配置日志，例如，测试环境中，将等级为 info 及以上的日志输出到文件；生产环境中，将等级为 error 及以上的日志发送邮件：

```python
# 测试环境
class TestingConfig(Config):
    ...
    @classmethod
    def init_app(cls, app):
        super().init_app(app)

        # 定义普通日志文件处理器
        file_handler = logging.FileHandler(cls.LOG_FILENAME, mode='a', encoding='utf-8', delay=False
        # 设置记录等级
        file_handler.setLevel(logging.INFOINFO)
        app.logger.addHandler(file_handler)

# 生产环境
class ProductionConfig(Config):
    ...
    @classmethod
    def init_app(cls, app):
        super().init_app(app)

        # 定义邮件日志处理器
```

```
        from logging.handlers import SMTPHandler

        credentials = None
        secure = None

        # 邮件各种参数和配置来自基类 config
        if getattr(cls, 'MAIL_USERNAME', None):
            credentials = (cls.MAIL_USERNAME, cls.MAIL_PASSWORD)
            if getattr(cls, 'MAIL_USE_TLS', None):
                secure = ()
        mail_handler = SMTPHandler(
            mailhost=(cls.MAIL_SERVER, cls.MAIL_PORT),
            fromaddr=cls.FLASKY_MAIL_SENDER,
            toaddrs=[cls.FLASKY_ADMIN],
            subject=cls.FLASKY_MAIL_SUBJECT_PREFIX + ' APPLICATION ERROR',
            credentials=credentials,
            secure=secure
        )
        # 设置发送等级
        mail_handler.setLevel(logging.ERROR)
        app.logger.addHandler(mail_handler)
```

这样在代码中用调用日志记录方法，根据不同环境，日志就能被恰当的处理了。

项目打包

一般我们会为一个项目创建一个文件夹，由于 Python 项目不需要编译，开发完成后，将项目文件夹拷贝到服务上就可以完成了部署。

在应用开发过程中，我们会陆续安装一些依赖库或模块，部署后，必须安装这些被依赖，应用才能运行，要记住安装了哪些依赖不是件轻松的事

幸好 `pip` 提供了导出依赖模块名录的功能，可以一并导出依赖名录：

- 将环境中依赖的外部模块名录导入到 requirements.txt 中

```
pip frerze > requirements.txt
```

- 在服务器上依据 requirements.txt 安装应用依赖

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

我们要将 requirements.txt 作为项目代码的一部分。

值得注意的是 `pip freeze` 命令并不是针对特定项目的，即，导出的是所在 Python 环境中的 所有外部模块。

如果一个 Python 环境中，创建了两个不同的项目，各自有不同的依赖，那么导出来的依赖会是两个项目以来的合集，虽然对部署来说没有有问题，但安装没必要的依赖不算是好事。

因此创建项目时，为其创建一个独立的 Python 虚拟环境是个好编程习惯。

Web 服务器

虽然 Flask 提供了内置 Web 服务器，但是那种服务器是为开发打造的，达不到生产环境中需要的安全和效率，细心的同学会注意到，用 `app.run()` 或者 `flask run` 启动应用时，都会得到一句警告：Do not use the development server in a production environment.

那么在生产环境中，需要用生产专用 Web 服务器，比如 uWSGI、Gunicorn、Gevent 等等，

> 注意：多数 Web 服务器只支持 Linux 操作系统，如果在 Windows 上部署，可以用 Apache + mod_wsgi 的方式

我们以 uWSGI 为例，了解下如何将 Flask 项目同 uWSGI 服务器关联。

安装 uWSGI

```
pip install uwsgi
```

指定启动脚本

启动脚本就是创建应用实例所在的代码文件。

如果没有明确的应用实例定义，例如用了工厂方法，就需要单独创建一个应用实例，例如，创建一个 `run.py` 脚本：

```
from myflask import create_app
import os

env_dist = os.environ
config = env_dist.get('MY_FLASK_ENV', 'default')
app = create_app(config)
```

- 从 `myflask` 项目中引入工厂方法 `create_app`
- 引入系统模块 `os`，为了读取环境变量
- 从环境变量 `MY_FLASK_ENV` 中读取 Flask 应用环境参数（环境变量名可随意），如果没有配置，默认值为 `default`，即为开发环境（具体配置参考：Flask 项目结构）
- 将 Flask 应用参数作为参数创建 Flask 应用实例

> 注意：在启动脚本中不要调用 `run` 方法，如果有需要放在 `if __name__ == '__main__'` 判断之下，否则 Web 服务器启动时，没创建一个应用实例（即运行一次启动脚本），就要启动一个 Flask 内置服务器，不仅浪费资源，还会出现端口冲突的错误

启动 uWSGI

```
uwsgi --http :9090 --wsgi-file run.py
```

- --http: 通过 http 可访问，绑定端口为 9090
- --wsgi-file：指定启动脚本

一般我们会将启动设置写在配置文件中，一是有方便加入更多配置，二是方便管理，uWSGI 支持 ini、xml、yaml、json 格式的配置文件

以 ini 格式为例，创建uwsgi.ini：

```
[uwsgi]
# 开启主进程
master = true
# 使用 http 协议，绑定 5000 端口
http=:5000
# 应用主目录
chdir = /home/alisx/justdopython/flaskapps
# 应用启动脚本
wsgi-file=/home/alisx/justdopython/flaskapps/run.py
# 启动脚本中定义的 Flask 实例 变量名
callable=app
# 应用使用 Python 虚拟环境路径
home=/home/alisx/justdopython/flaskapps/.venv
# Web 服务器工作进程数
processes=4
# 每个进程中线程数
threads=2
# uWSGI 进程号存放文件，用户停止和关闭
pidfile =/home/alisx/justdopython/flaskapps/uwsgi.pid
```

请注意 `home` 配置项，用来指定 Python 虚拟环境

启动 uWSGI：

```
uwsgi uwsgi.ini
```

uWSGI 功能强大，配置丰富，这里只展示了基本的配置参数，想要了解更多可以参考文后 uWSGI 参考链接。

前置服务器

> 如果只是在服务器上部署一个 Flask 应用，可以跳过这里，用时再看不迟

尽管 uWSGI 功能强，性能高，完全可以胜任 Web 服务器，在实际部署中，我们还是想将其放在功能更全性能更好的专职前置 Web 服务器之后

这样做的好处是：

- 可以部署多个 Web 应用
- 不必争夺 80 端口
- 方便配置高并发和负载均衡

我们以现在流行从前置（反向代理）服务器 nginx 为例作

如果服务器上没有 nginx 需要先安装，以 ubuntu 为例：

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

在 nginx 的虚拟服务器的 location 中指定后端服务器：

```
...
location / {
    include uwsgi_params;
    uwsgi_pass 127.0.0.1:9090;
}
...
```

- include: 让 nginx 加载 uwsgi 功能模块
- uwsgi_pass: 指定用 uwsgi 协议的后端应用的地址和端口，此时 uWSGI 启动参数 http 要换位 socket: `uwsgi --socket 127.0.0.1:9090`

最后重启 nginx 就可以了向外提供我们的 Web 服务了。

关于 uWSGI 通信协议

前面 nginx 配置中，使用了 `uwsgi_pass` 指定后端服务器和通信协议

对于 uWSGI 而言，既可以指 Web 服务器，也可以指 uwsgi 通信协议（类似于 http 协议），是uWSGI 服务器其的默认通信协议，具有更好的性能和安全性，启动 uWSGI 服务器时，可以指定所使用的协议：

- `--http` : 会启动一个 http 进程，来接受 http 请求，该进程地位等同于 nginx，相当于前置服务器，http 进程使用 uwsgi 协议与 后端服务器通信
- `--http-socket` : 不启动 http 进程，需要前置服务器，且前置服务器不支持 uwsgi 协议的情况下使用
- `--socket` : 前置服务器支持 uwsgi 协议情况使用，例如用 nginx 作为前置服务器时

总结

今天介绍了如何部署一个 Flask 应用，从部署前的准备，一直到前置服务器的配置，其中以 uWSGI Web 服务器为例，介绍了如何将 Flask 应用绑定到生产服务器上，以及一些前置服务器的基本知识，希望对您发布自己的应用有一些帮助。

最后，示例代码中提供了较为完整的例子，特别是通过 config 配置日志的部分，请您参考。

参考

- https://uwsgi-docs.readthedocs.io/en/latest/WSGIquickstart.html
- https://www.cnblogs.com/pengyusong/p/5780251.html
- http://www.ruanyifeng.com/blog/2016/03/systemd-tutorial-commands.html

> 示例代码：Python-100-days-day130

PS: 公号内回复「Python」即可进入 Python 新手学习交流群，一起 **100天计划！**

-END-

Python 技术

关于 Python 都在这里