Flask 常用设计结构

在小型Flask项目中, 一般只需要一个文件或者几个文件即可

单文件Flask程序

```python
# 只有一个文件: app.py

from flask import Flask


app = Flask(__name__)


@app.route('/')
def index():
    return 'Hello World!'


if __name__ == '__main__':
    app.run()

```

几个文件的Flask程序

```
.
└── app
    ├── __init__.py
    ├── commands.py
    ├── data
    ├── errors.py
    ├── forms.py
    ├── models.py
    ├── static
    │   ├── css
    │   └── js
    ├── templates
    ├── utils.py
    └── views.py

```

在大型Flask项目中, 主要有三种常见的项目组织架构:

- 功能式架构
- 分区式架构

- 混合式架构

功能式架构

在功能式架构中, 程序包由各个代表程序组件(功能)的子包组成, 比如blueprints(), forms(), templates(), models()等, 在这些子包中, 按照程序等板块分模块来组织代码, 比如forms子包下包含`front.py`,`auth.py`,和`dashboard.py`,这种架构结构清晰, 更容易在开发时让开发者迅速找到文件, 其他维护者也能迅速了解程序结构. 使用功能式架构的程序包目录如下所示:

```
.
└── app
    ├── __init__.py
    ├── blueprints
    │   ├── __init__.py
    │   ├── auth.py
    │   ├── dashboard.py
    │   └── front.py
    ├── forms
    │   ├── __init__.py
    │   ├── auth.py
    │   ├── dashboard.py
    │   └── front.py
    ├── static
    │   ├── css
    │   ├── images
    │   └── js
    └── templates
        ├── auth
        ├── base.html
        ├── dashboard
        └── front
```

因为程序比较简单, 所以蓝本主要是用来组织路由, 所以项目中等蓝本直接在blueprints包下的模块中创建, 如果蓝本需要注册更多处理程序, 比如错误处理函数, 请求处理函数等, 可以在blueprints包中为每个蓝本创建子包, 目录结构如下:

```
.
└── app
    ├── __init__.py
    ├── blueprints
    │   ├── __init__.py
    │   ├── auth
    │   │   ├── __init__.py
    │   │   ├── errors.py
    │   │   └── views.py
    │   ├── dashboard
    │   │   ├── __init__.py
    │   │   ├── errors.py
    │   │   └── views.py
    │   └── front
    │       ├── __init__.py
```

```
    |··      ├── errors.py
    |··      └── views.py
    ├── forms
    |·· ├── __init__.py
    |·· ├── auth.py
    |·· ├── dashboard.py
    |·· └── front.py
    ├── static
    |·· ├── css
    |·· ├── images
    |·· └── js
    └── templates
        ├── auth
        ├── base.html
        ├── dashboard
        └── front
```

通过蓝本创建子包还可以支持为蓝本创建独立的templates和static文件夹

```
.
└── app
    ├── __init__.py
    ├── blueprints
    |·· ├── __init__.py
    |·· ├── auth
    |·· |·· ├── __init__.py
    |·· |·· ├── errors.py
    |·· |·· ├── static
    |·· |·· ├── templates
    |·· |·· └── views.py
    |·· ├── dashboard
    |·· |·· ├── __init__.py
    |·· |·· ├── errors.py
    |·· |·· ├── static
    |·· |·· ├── templates
    |·· |·· └── views.py
    |·· └── front
    |··      ├── __init__.py
    |··      ├── errors.py
    |··      ├── static
    |··      ├── templates
    |··      └── views.py
    └── forms
        ├── __init__.py
        ├── auth.py
        ├── dashboard.py
        └── front.py
```

和在单模块中创建蓝本不同, 当在子包中创建蓝本时, 为了方便其他模块导入蓝本对象, 这时蓝本对象在蓝本子包的`__init__.py`中构建. 而且因为蓝本在构造文件中定义, 为了把路由、错误处理器、请求处理函数等和蓝本对象关联起来, 需要在`__init__.py`中导入这些模块. 为了避免循环依赖, 在`__init__.py`文件底部添加这些导入语句, 如下所示:

```
# app/blueprints/front/__init__.py
```

```
2
3 from flask import Blueprint
4 front_bp = Blueprint('front', __name__)
5
6 from app.blueprints.front import views, errors
```

在路由模块等要使用蓝本对象的地方可以直接导入这里创建等蓝本对象

```
1 # app/blueprints/front/views.py
2
3 from app.blueprints.front import front_bp
4
```

分区式架构

在分区式架构中, 程序被按照自身板块分成不同的子包, app使用分区式架构可以可以分别创建`front`、`auth`、和`dashboard`三个子包, 这些子包直接在程序包的根目录下创建, 子包中使用模块组织不同的程序组件, 比如`views.py`、`forms.py`等. 这种分类自然决定了每一个子包都对应着一个蓝本, 这时蓝本在每个子包的`__init__.py`中创建. 使用分区式架构的程序包目录结构示例如下:

```
1 .
2 └── app
3     ├── __init__.py
4     ├── auth
5     │   ├── __init__.py
6     │   ├── forms.py
7     │   ├── static
8     │   ├── templates
9     │   └── views.py
10    ├── dashboard
11    │   ├── __init__.py
12    │   ├── forms.py
13    │   ├── static
14    │   ├── templates
15    │   └── views.py
16    └── front
17        ├── __init__.py
18        ├── forms.py
19        ├── static
20        ├── templates
21        └── views.py
```

混合式架构

混合式架构, 顾名思义, 就是不按照常规分类来组织. 比如, 采用类似分区式架构的子包来组织程序, 但各个蓝本共用程序包根目录下的模版文件夹和静态文件夹:

```
1 .
2 └── app
```

```
    ├── __init__.py
    ├── auth
    │   ├── __init__.py
    │   ├── forms.py
    │   └── views.py
    ├── dashboard
    │   ├── __init__.py
    │   ├── forms.py
    │   └── views.py
    ├── front
    │   ├── __init__.py
    │   ├── forms.py
    │   └── views.py
    ├── static
    └── templates
```

或是某个蓝本采用分区式架构单独组织, 其他蓝本则使用功能式架构统一放到blueprints子包中.

如何选择

不同类型等程序适合不同的组织方式, 一般来说, 如果如果程序各个功能之间联系较为紧密, 我们可以采用功能式组织方式. 反之, 则适合采用分区式.