# 课程介绍

* 1. flask-script插件使用;（了解）
* 2. Flask蓝图;（掌握）
* 3. Flask模板JinJa2;（了解）
* 4. flask-bootstrap插件；（了解）

1. Flask-script插件使用
   1. 使用场景

类似于Django开发那样，如果要使用类似数据库的迁移，增加交互调试命令的话，Flask原生是不支持的，所以为了能够让后续开发方便，我们先使用一个叫做Flask-script的插件，他可以扩展Flask的功能，支持命令交互模式并且添加自定义命令。

Flask-Script 是一个 Flask 扩展，为 Flask 程序添加了一个命令行解析器。Flask-Script 自带了一组常用选项，而且还支持自定义命令。

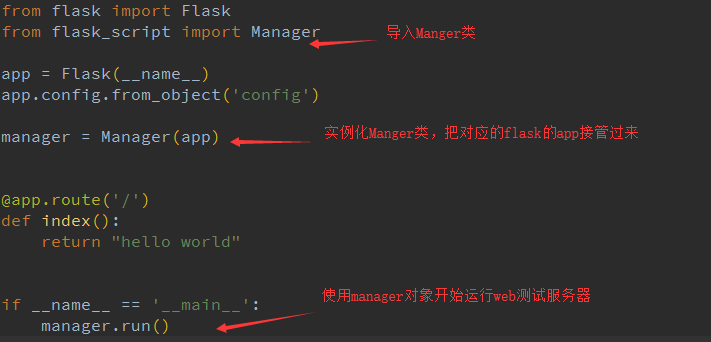
* 1. 安装和使用
     1. 安装flask-script插件

pip install flask-script

保证在虚拟环境下敲入该命令。

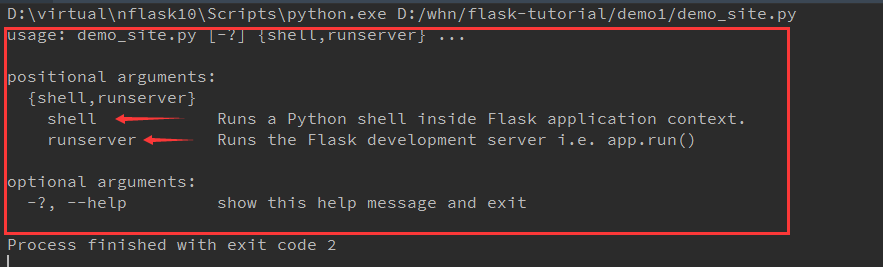
* + 1. 使用插件，扩展原app

把原来的flask程序修改为：



* + 1. 测试效果

此时再运行该文件后，会看到下面的结果：



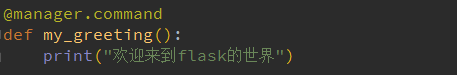
shell 命令用于在程序的上下文中启动 Python shell 会话。你可以使用这个会话中运行维护任务或测试，还可调试异常。

runserver 命令用来启动 Web 服务器。运行 python demo\_site.py runserver 将以运行Flask自带的测试Web服务器。

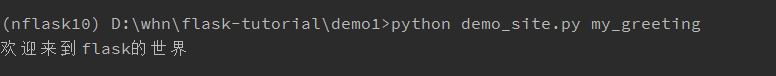
使用 —help 可以查看到相关的帮助信息。

* 1. 添加命令
     1. command装饰器法

使用Manager实例化的对象提供了一个command装饰器，将把装饰的函数名作为命令，添加到交互模式上。

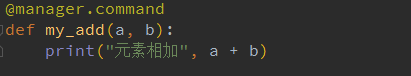


此时再执行



将看到效果。

再看一个带参数的例子：

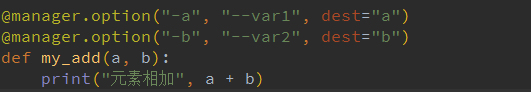


使用的时候：



* + 1. 使用option功能进行精细化控制（扩展）

通过option的dest参数，将用户输入的值指向参数指定名称





1. Flask蓝图
   1. 业务分离管理

前面我们的业务代码全部写到了一个文件，小型网站还好，如果业务层面稍微多些，非常不适合团队开发。

* + 1. 视图函数的分离问题

将视图函数保存到一个单独目录下的单独文件里。

但这个时候，会发现几个问题：

* 视图函数注册路由的app，需要导入主模块的app， 但主模块怎么导入视图文件里的函数名；
* 一旦在主模块里导入视图函数，就出现了循环导入的问题了；
  + 1. 解决方案

在Flask中提供了一个叫做蓝图（**BluePrint**）的模块。

蓝图和app类似，也可以定义路由。不同的是，在蓝图中定义的路由处于休眠状态，直到蓝图注册到程序上后，路由才真正成为程序的一部分。

每个子业务都配置一个蓝图，然后最后把所有蓝图都注册到主app上。

* 1. 使用蓝图
     1. 在子系统中申请蓝图

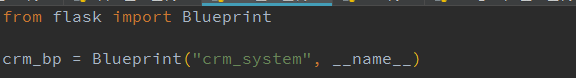
导入蓝图类：

from flask import Blueprint

注意是首字母大写的类。

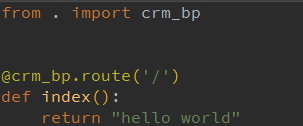
通过实例化一个 Blueprint 类对象可以创建蓝本。这个构造函数有两个必须指定的参数：蓝本的名字和蓝本所在的包或模块。

蓝图建议申请在子系统包的\_\_init\_\_.py文件上。

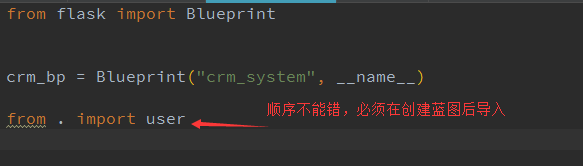


* + 1. 在子系统的视图模块中引入蓝图

由于子系统的视图模块就在这个包里，不存在循环引用的问题。



* + 1. 在子系统包中导入视图模块



* 1. 蓝图的使用技巧

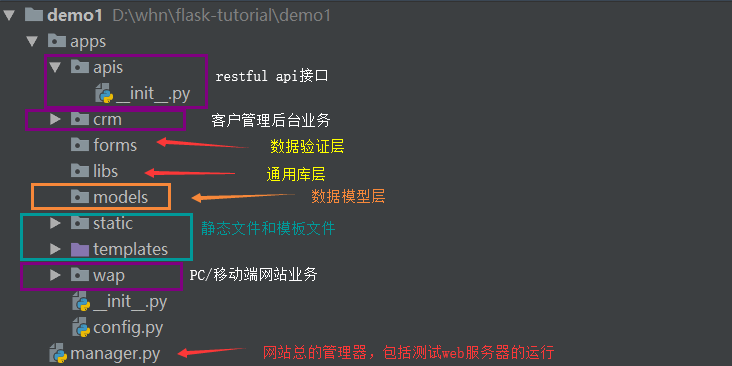
蓝图把视图函数和路由从主程序中分离出来了，一般分离的原则是根据业务URL或者业务主体的不同而分离，比如/books/...算一个业务，/movie/...算一个业务，/admin/...算一个业务，这样，对于每个业务下面的路由视图函数注册时，都需要把这样的前缀写全，非常麻烦。

蓝图提供了一个url\_prefix，同时注册时也有这个参数，不过注册的优先级最高。

app.register\_blueprint*(*crm\_bp, url\_prefix='/crm1'*)*

* 1. 总结

一个典型的Flask工程结构目录



1. Flask的模板引擎

Flask 使用了一个名为 Jinja2 的强大模板引擎，他模仿Django的模板引擎，所以对于有Django模板DTL语法经验的话，学习Jinja2会非常容易的。

模板语法主要分为两种：

变量，所有的变量使用{{ var }}形式，由视图函数传递给模板。

标签，所有的标签使用{% tag %}形式，有控制逻辑，使用外部表达式，宏定义等。

* 1. 模板查找路径及渲染方法

默认情况下，Flask 在app注册的同级目录下的 templates 文件夹中寻找模板。

Flask提供的render\_template函数把Jinja2模板引擎集成到了程序中。

render\_template 函数的第一个参数是模板的文件名。随后的参数都是键值对，表示模板中变量对应的真实值。

他跟Django的区别是，Django传递的是一个字典结构，而Flask提供的是命名参数。

* 1. 变量

Jinja2 能识别所有类型的变量，甚至是一些复杂的类型，例如列表、字典和对象。在模板中使用变量的一些示例如下：

<p>A value from a dictionary: {{ mydict['key'] }}.</p>

<p>A value from a list: {{ mylist[3] }}.</p>

<p>A value from a list, with a variable index: {{ mylist[myintvar] }}.</p>

<p>A value from an object's method: {{ myobj.somemethod() }}.</p>

他可以使用Python类似的语法进行访问。

可以使用过滤器修改变量，过滤器名添加在变量名之后，中间使用竖线分隔。

|  |  |
| --- | --- |
| **safe** | **渲染值时不转义** |
| **capitalize** | 把值的首字母转换成大写，其他字母转换成小写 |
| **lower** | 把值转换成小写形式 |
| **upper** | 把值转换成大写形式 |
| **title** | 把值中每个单词的首字母都转换成大写 |
| **trim** | 把值的首尾空格去掉 |
| **striptags** | 渲染之前把值中所有的 HTML 标签都删掉 |

过滤器参数使用函数调用语法而不是用冒号分 隔过滤器名和参数。比如default的过滤器使用。

更多过滤器可以参考文档：

http://docs.jinkan.org/docs/jinja2/templates.html#builtin-filters

这里列出一些常用过滤器：

abs(value)：返回一个数值的绝对值。 例如：-1|abs。

default(value,default\_value,boolean=false)：如果当前变量没有值，则会使用参数中的值来代替。

escape(value)或e：转义字符，会将<、>等符号转义成HTML中的符号。例如：content|escape或content|e。

first(value)：返回一个序列的第一个元素。names|first。

format(value,\*arags,\*\*kwargs)：格式化字符串。例如以下代码：

{{ "%s" - "%s"|format('Hello?',"Foo!") }}将输出：Helloo? - Foo!

last(value)：返回一个序列的最后一个元素。示例：names|last。

length(value)：返回一个序列或者字典的长度。示例：names|length。

join(value,d=u'')：将一个序列用d这个参数的值拼接成字符串。

safe(value)：如果开启了全局转义，那么safe过滤器会将变量关掉转义。示例：content\_html|safe。

int(value)：将值转换为int类型。

float(value)：将值转换为float类型。

lower(value)：将字符串转换为小写。

upper(value)：将字符串转换为小写。

replace(value,old,new)： 替换将old替换为new的字符串。

truncate(value,length=255,killwords=False)：截取length长度的字符串。

striptags(value)：删除字符串中所有的HTML标签，如果出现多个空格，将替换成一个空格。

trim：截取字符串前面和后面的空白字符。

string(value)：将变量转换成字符串。

wordcount(s)：计算一个长字符串中单词的个数。

* 1. 标签
     1. for循环结构

目的类似，但语法和Django有些不同，具体参考：

http://docs.jinkan.org/docs/jinja2/templates.html#for

|  |  |
| --- | --- |
| **变量** | **描述** |
| **loop.index** | 当前循环迭代的次数（从 1 开始） |
| **loop.index0** | 当前循环迭代的次数（从 0 开始） |
| **loop.revindex** | 到循环结束需要迭代的次数（从 1 开始） |
| **loop.revindex0** | 到循环结束需要迭代的次数（从 0 开始） |
| **loop.first** | 如果是第一次迭代，为 True 。 |
| **loop.last** | 如果是最后一次迭代，为 True 。 |
| **loop.length** | 序列中的项目数。 |
| **loop.cycle** | 在一串序列间期取值的辅助函数。 |

在一个 for 循环块中你可以访问这些特殊的变量:

* + 1. if结构

{% if user %}

Hello, {{ user }}!

{% else %}

Hello, Stranger!

{% endif %}

* + 1. 宏结构

Jinja2 还支持宏。宏类似于 Python 代码中的函数。例如：

{% macro render\_comment(comment) %}

<li>{{ comment }}</li>

{% endmacro %}

<ul>

{% for comment in comments %}

{{ render\_comment(comment) }}

{% endfor %}

</ul>

为了重复使用宏，我们可以将其保存在单独的文件中，然后在需要使用的模板中导入：

{% import 'macros.html' as macros %}

<ul>

{% for comment in comments %}

{{ macros.render\_comment(comment) }}

{% endfor %}

</ul>

* + 1. include和extends标签

这2个标签和django的类似。

1. Flask特有变量和函数
   1. Flask路由和静态文件管理

不要直接在模板中写死静态文件或路由的路径，应该使用url\_for的方式生成路径。在Jinja2模板中可以直接使用该函数：

{{ url\_for(‘index’) }}，产生index的endpoint的URL值。

{{ url\_for(‘static’, filename=’img/a.jpg’)，产生’/static/img/a.jpg’

* 1. config

可以在模板中直接访问Flask当前的config对象：

{{ config.DEFINE\_VAR }}

* 1. request

代表Flask中代表当前请求的request对象：

{{ request.url }}

1. 课程总结
   1. 重点
      * 1. flask的蓝图使用
        2. 静态文件和路由产生的方法
   2. 难点
      * 1. 工程目录结构
        2. 循环引用的问题
2. 课后练习
   * + 1. 写完课堂上的代码
       2. 自学《Flaskweb开发》电子书的3.2 3.3，熟悉bootstrap插件的使用。
3. 面试题
   * + 1. Django的路由和Flask的路由区别
4. 扩展知识或课外阅读推荐（可选）