# Flask Web 开发实战：入门、进阶与原理解析 (李辉) (Z-Library)

# 初识Flask

## 搭建开发环境

Flask 是使用Python 编写的Web 微框架。Web 框架可以让我们不用关心底层的请求响应处理， 更方便高效地编写Web 程序。因为Flask 核心简单且易千扩展，所以被称作微框架(microframework)。

Flask 有两个主要依赖， 一个是WSGI (Web Server Gateway Interface, Web 服务器网关接口）工具集一Werkzeug (http://werkzeug.pocoo.org/ ），另一个是Jinja2 模板引擎( http: //jinjapocoo.org/ ） 。

打开命令行窗口，切换到合适的目录， 然后使用下面的命令把本书的示例程序仓库复制到本地，并切换进项目根目录：

$ git clone https : //github . com/greyli / helloflask .git

$ cd helloflask

或者在HelloFlask 的GItHub 页面（ https.I/gIthubcom/greyh/helloflask) 单击了Fork 按

丁钮，那么可以使用你自己的GitHub 用户名未替换掉上面的greyli, 这将复制一份派生仓

库，你可以自由地修改和提交代码。

### Pipenv工作流

Pipenv 是基于pip 的Python 包管理工具，它和pip 的用法非常相似，可以看作pip 的加强

版，它的出现解决了旧的pip+virtual en v+requ i rem en ts. txt 的工作方式的弊端。具体来说，它是pip 、和pfile 和Virtualenv 的结合体，它让包安装、包依赖管理和虚拟环境管理更加方便，使用它可以实现高效的Python 项目开发工作流。

1. **安装pip和pipenv**

查看pip是否安装 pip --version

安装指定的包 pip install 包名称

使用pip安装pipenv pip install pipenv

查看pipenv是否安装 pipenv --version

1. **创建虚拟环境**

在Python 中，虚拟环境(virtual enviroment) 就是隔离的Python 解释器环境。通过创建虚拟环境，你可以拥有一个独立的Python 解释器环境。这样做的好处是可以为每一个项目创建独立的Python 解释器环境，因为不同的项目常常会依赖不同版本的库或Python 版本。使用虚拟环境可以保待全局Python 解释器环境的干净，避免包和版本的混乱，并且可以方便地区分和记录每个项目的依赖，以便在新环境下复现依赖环境。

虚拟环境通常使用Virtualenv 来创建，但是为了更方便地管理虚拟环境和依赖包，我们将会使用集成了Virtualenv 的Pipenv 。

确保我们当前工作目录在示例程序项目的根目录，即helloflask 文件夹中， 然后使用pipenv insta ll 命令为当前的项目创建虚拟环境：

为当前项目创建虚拟环境 pipenv install

这会为当前项目创建一个文件夹，其中包含隔离的Python 解释器环境，并且安装pip 、

wheel 、setuptools 等基本的包

默认情况下，和penv 会统一管理所有虚拟环境。在Win!o\_ws 系统中，虚拟环境文件夫会在C: \U sers\Administrator\. virtualenvs\ 目录下创建，而L i nux 或macOS 会在～/ .local/share/virtual envs/ 目录下创建。如果你想在项目目录内创建虚拟环境文件夫，可以设置环境变量PIPENV\_ VENV IN\_PROJECT, 这时名为．venv 的虚拟环境文件夹将在项目根目录被创建。

虚拟环境文件夹的目录名称的形式为“当前项目目录名＋ 一串随机字符＂，比如helloflask-5PaOZfZw

显式激活当前项目的虚拟环境 pipenv shell

不显式激活虚拟环境时，执行命令 pipenv run python hello.py

1. **管理依赖**

在创建虚拟环境时，如果项目根目录下没有和pfile 文件， pip env install 命令还会在项目文件夹根目录下创建和pfile 和Pipfile.lock 文件，前者用来记录项目依赖包列表，而后者记录了固定版本的详细依赖包列表。当我们使用Pipenv 安装／删除／更新依赖包时， 和pfile 以及Pipfile.lock 会自动更新。

查看当前环境下的依赖情况 pipenv graph

在虚拟环境中查看依赖情况 pip list

### 安装Flask

在创建好的虚拟环境中安装Flask pipenv install flask

Pipenv 会自动帮我们管理虚拟环境，所以在执行pip env in s ta ll 安装Pytho n 包时，无论是否激活虚拟环境，包都会安装到虚拟环境中。后面我们都将使用和p en v 安装包，这相当于在激活虚拟环境的情况下使用pip 安装包。只有需要在全局环境下安装／ 更新／ 删除包，我们才会使用pip

更新包 pipenv update flask(不写包名称 则更新该环境中的所有包)

### 集成开发环境(Pycharm)

## Hello,Flask!

切换到示例程序所在目录 cd demo/flask(目录路径)

对于简单的程序来说，程序的主模块一般命令为app.py 。你也可以使用其他名称，比如丁hello.py , 但是要避免使用flask.py, 因为这和F lask 本身冲突。

### 创建程序实例

从flask保重导入Flask类，并实例化为app

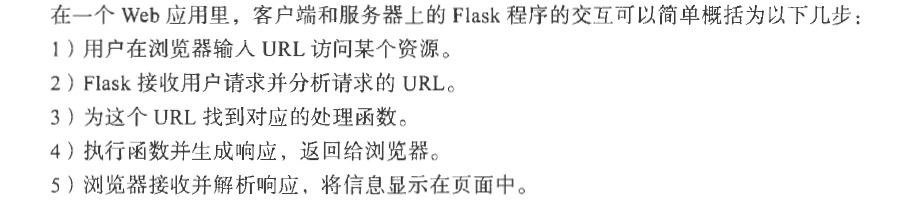
from flask import Flask

app = Flask ( \_ \_name\_ \_ )

传入Flask 类构造方法的第一个参数是模块或包的名称，我们应该使用特殊变量—name—°

Python 会根据所处的模块来赋予—name— 变量相应的值，对千我们的程序来说(app .py) ，这个值为app 。除此之外，这也会帮助Flask 在相应的文件夹里找到需要的资源，比如模板和静态文件。

### 注册路由



在上面这些步骤中，大部分都由F lask 完成，我们要做的只是建立处理请求的函数，并为其

定义对应的URL 规则。只需为函数附加app.route(） 装饰器，并传入URL 规则作为参数，我们就可以让URL 与函数建立关联。这个过程我们称为**注册路由(route)** ，

路由负责管理URL 和函数之间的映射， 而这个函数则被称为视图函数(view function) 。

（注册路由：）定义视图函数，并将其和装饰器绑定起来；

@app.route(‘/’) #绑定装饰器

def index() : #视图函数

return'<hl>Hello, World! </ hl> '

route(） 装饰器的第一个参数是URL 规则，用字符串表示，必须以针杠（/) 开始。这里的URL 是相对URL （又称为内部URL) ，即不包含域名的URL 。以域名www.belloflaskcom 为例， “ /" 对应的是根地址（ 即www.hellofl ask . com ) ，如果把URL 规则改为“ /hello " ，则实际的绝对地址（外部地址）是www.belloflask.com/bello 。

1. **为视图函数绑定多个URL**

一个视图函数可以绑定多个URL, 比如下面的代码把／hi 和/he ll o 都绑定到say\_h e llo(） 函数上，这就会为say\_hello 视图注册两个路由，用户访问这两个URL 均会触发say\_hello() 函数，获得相同的响应。

@app.route('/hi')

@app.route('/hello')

def say\_hello():

return '<h1>Hello, Flask!</h1>'

1. **动态URL**

我们不仅可以为视图函数绑定多个URL, 还可以在URL 规则中添加变盐部分，使用＂ ＜变

量名＞＂ 的形式表示。F l ask 处理请求时会把变量传入视图函数，所以我们可以添加参数获取这个变最值。代码清单1-3 中的视图函数greet(），它的URL 规则包含一个name 变量。

@app.route('/greet', defaults={'name': 'Programmer'}) # 设置默认变量名

@app.route('/greet/<name>')

def greet(name):

return '<h1>Hello, %s!</h1>' % name

## 启动开发服务器

Flask 内置了一个简单的开发服务器（由依赖包Werkzeug 提供），足够在开发和测试阶段

使用。

### Run flask!

Flask 通过依赖包Click 内置了一个CLI ( Command Line Interface , 命令行交互界面）系统。

当我们安装Flask 后，会自动添加一个flask 命令脚本， 我们可以通过flask 命令执行内置命令、扩展提供的命令或是我们自己定义的命令

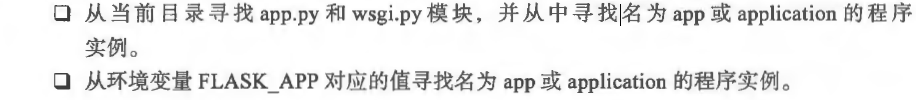
启动内置的开.服务器 flask run

确保执行命令前激活了虚拟环境~ pipenv shell ) ，否则需要使用pipenv run flask run 命令启动开发服务器

1. **自动发现程序实例**

一般来说，在执行flask run 命令运行程序前，我们需要提供程序实例所在模块的位置。我

们在上面可以直接运行程序，是因为Flask 会自动探测程序实例，自动探测存在下面这些规则：



因为我们的程序主模块命名为app .py , 所以flask run 命令会自动在其中寻找程序实例。如

果你的程序主模块是其他名称， 比如hello.py , 那么需要设置环境变掀FLASK\_APP, 将包含程序实例的模块名赋值给这个变量。

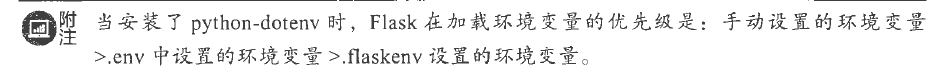
在Wi nd ows 系统中使用set 命令设置环境变量：

set FLASK APP=hello

1. **管理环境变量**

Flask 的自动发现程序实例机制还有第三条规则：如果安装了python-dotenv, 那么在使用

flask run 或其他命令时会使用它自动从.flaskenv 文件和.env 文件中加载环境变量。



安装环境变量管理工具到虚拟环境中 pipenv install python-dotenv

在项目根目录下分别创建两个文件： .env 和．flaskenv。

. flaskenv 用来存储和Flask 相关的公开环境变量，比如FLASK\_APP ; 而.env 用来存储包含敏感信息的环境变量，比如后面我们会用来配置Email 服务器的账户名与密码。

在.fl askenv 或e nv 文件中，环境变量使用键值对的形式定义，每行一个，以＃开头的为注释，如下所示：

SOME VAR=1

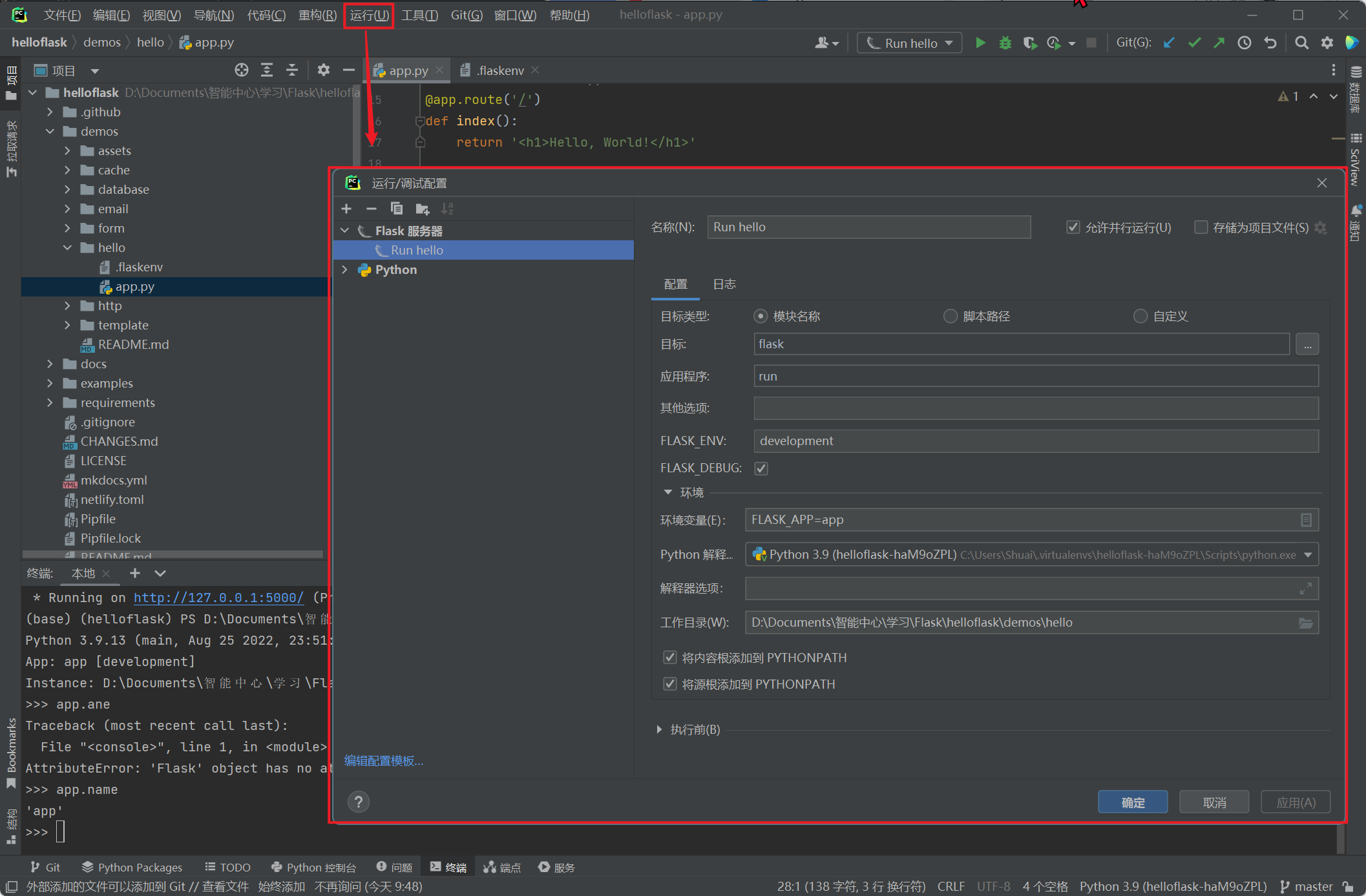
＃这是注释

FOO= "BAR"

.env 包含敏感信息，除非是私有项目，否则绝对不能提交到Git 仓库中。当你开发一个新项目时，记得把它的名称添加到gitignore 文件中，这会告诉G it 忽略这个文件。gitignore 文件是一个名为.gitign ore 的文本文件，它存储了项目中G it 提交时的忽略文件规则清单。

Pyth on 项目的g 山gnore 模板可以参考http s: //g ithub .co m/github/g 山gnore/blob/mas ter/Python .g 山gnore 。使用PyC harm 编写程序时会产生一些配置文件，这些文件保存在项目根目录下的idea 目录下，关于这些文件的忽略设置可以参考https://www.gi tignore. io/ api /p yc harm 。

1. **使用Pycharm运行服务器**



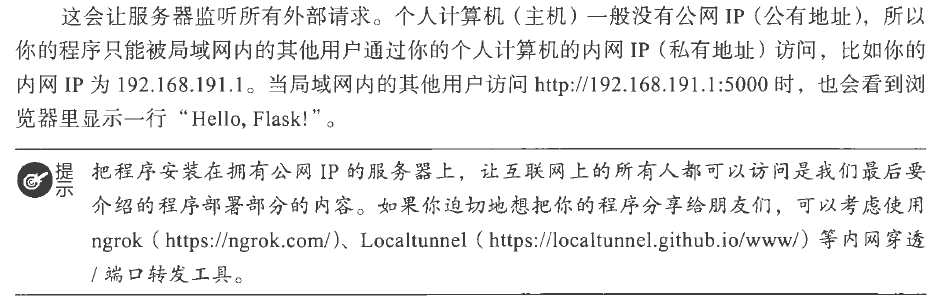


### 其他的启动设置

1. **使服务器外部可见**

我们在上面启动的Web 服务器默认是对外不可见的，可以在run 命令后添加－－host 选项将主机地址设为0.0.0.0 使其对外可见：

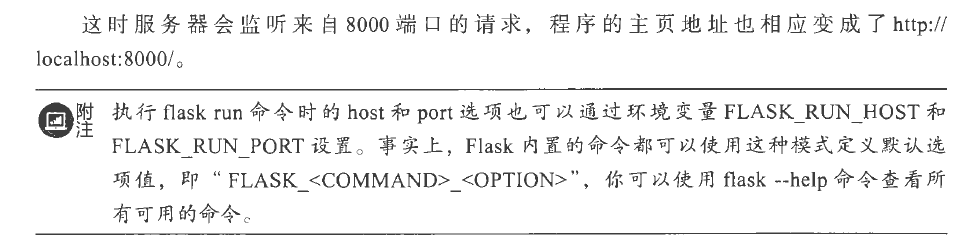
flask run - - host==0.0.0.0



1. **改变默认端口**

Flask 提供的Web 服务器默认监听5000 端口，你可以在启动时传入参数来改变它：

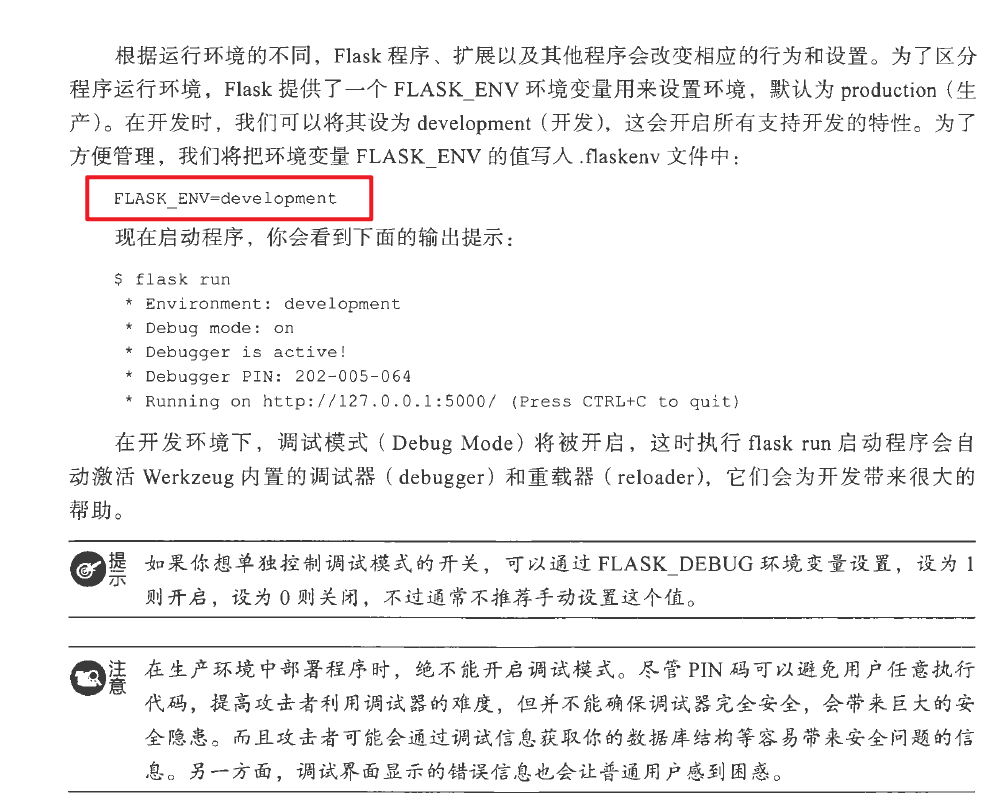
flask run --port=8000



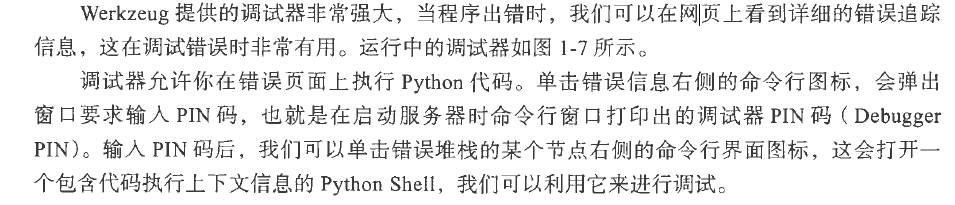
### 设置运行环境

**开发环境**( development e nviroment ) 和**生产环境**(production enviroment ) 是我们后面会频

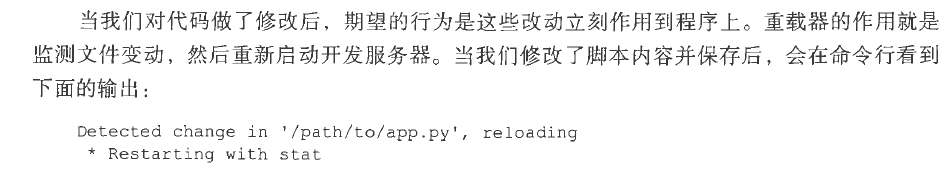
繁接触到的概念。开发环境是指我们在本地编写和测试程序时的计算机环境，而生产环境与开发环境相对，它指的是网站部署上线供用户访问时的服务器环境。

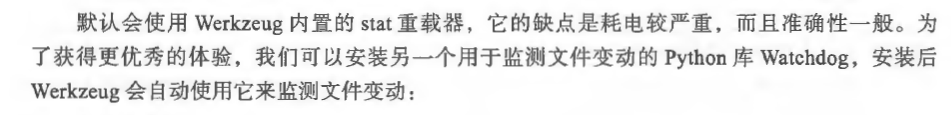


1. **调试器**

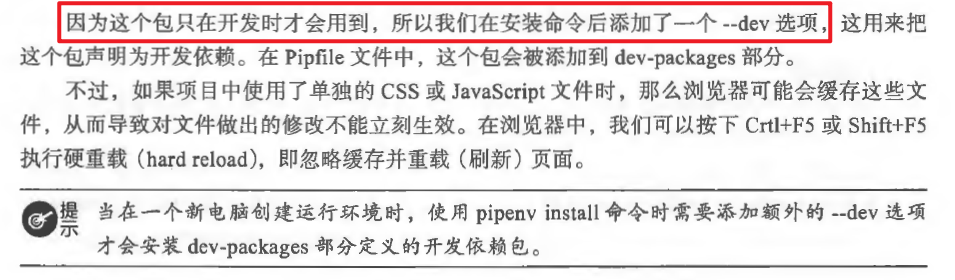


1. **重载器**

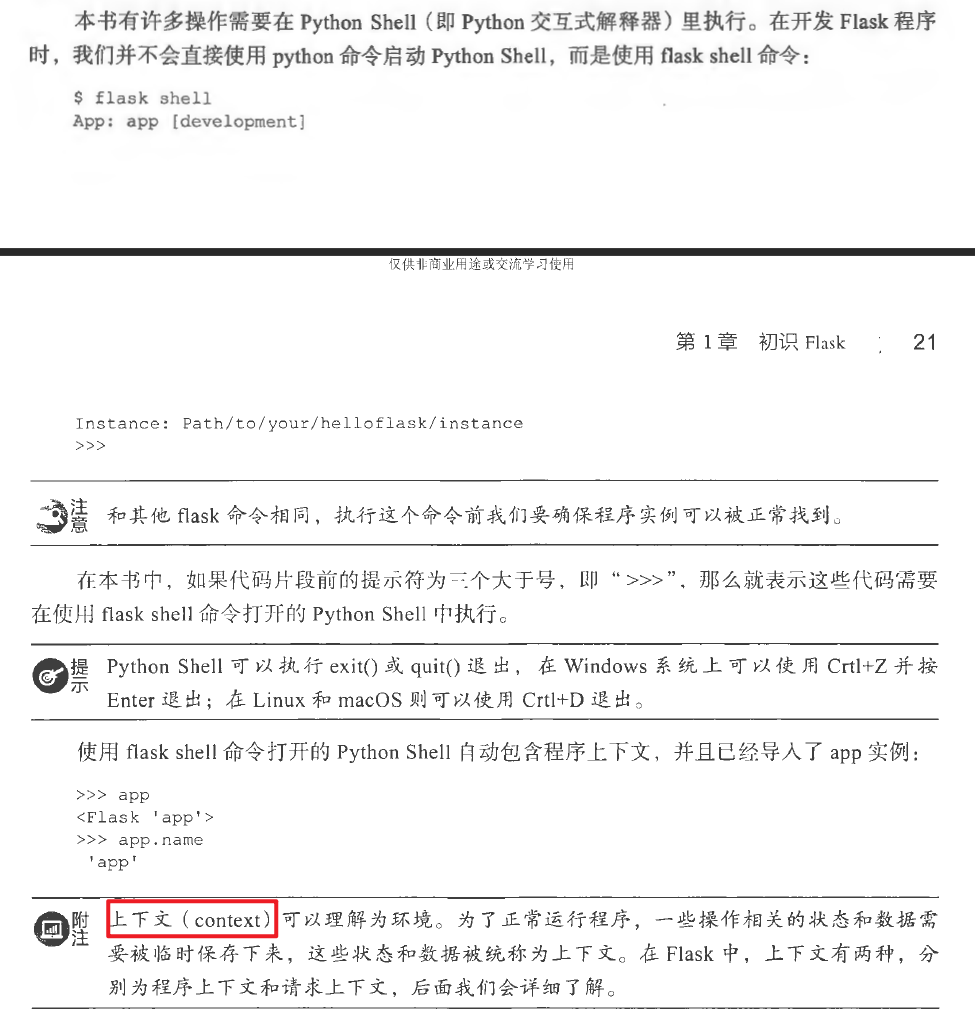




Pipenv install watchdog --dev



## Python Shell



## Flask扩展

**扩展( extension )** 即使用Flask 提供的API 接口编写的Python 库，可以为F lask 程序添加各种各样的功能。

大部分Flask 扩展用来集成其他库，作为F l ask 和其他库之间的薄薄一层胶水。因为Flask 扩展的编写有一些约定，所以初始化的过程大致相似。大部分扩展都会提供一个扩展类，实例化这个类，并传入我们创建的程序实例app作为参数，即可完成初始化过程。通常，扩展会在传入的程序实例上注册一些处理函数，并加载一些配置。

以某扩展实现了Foo 功能为例，这个**扩展的名称**将是Flask- Foo 或Foo-Flask ;

**程序包或模块的命名**使用小写加下划线，即flask\_foo （即导入时的名称）；

用于初始化的类一般为Foo, 实例化的类实例一般使用小写，即foo。

初始化这个假想中的Flask-Foo 扩展的示例如下所示：

from flask import Flask

from flask\_foo import Foo

app = Flask(name )

foo = Foo(app)

## 项目配置

在很多情况下，你需要设置程序的某些行为，这时你就需要使用**配置变量**。在F lask 中，**配**

**置变量**就是一些大写形式的Python 变量，你也可以称之为配置参数或配置键。

使用统一的配置变量可以避免在程序中以硬编码( hard coded) 的形式设置程序。

在一个项目中，你会用到许多配置： **Flask 提供的配置**， **扩展提供的配置**，还有**程序特定的配置**。

和平时使用变量不同，这些配置变量都通过Flask 对象的app . config 属性作为统一的接

口来设置和获取，它指向的Config 类实际上是字典的子类，所以你可以像操作其他字典一样操作它。

Flask 提供了很多种方式来加载配置。

1. **设置单个配置**

比如，你可以像在字典中添加一个键值对一样来**设置一个配置**：

app . conf ig [ ' ADM IN\_ NAME'] ='Peter '

1. **设置多个配置**

使用update()方法则可以一次加载多个值：

app . config.update(

TESTING=True ,

SECRET\_KEY='\_S#yF4Q8z\n\xec]/'

)

1. **通过文件导入配置**

除此之外，你还可以把配置变擞存储在单独的Python 脚本、JSON 格式的文件或是Python

类中， config 对象提供了相应的方法来导入配置，具体我们会在后面了解。

和操作字典一样，读取一个配置就是从confi g 字典里通过将配置变量的名称作为键读取对

应的值：value = app . config[ ' ADMIN\_NAME ' ]

## URL与端点

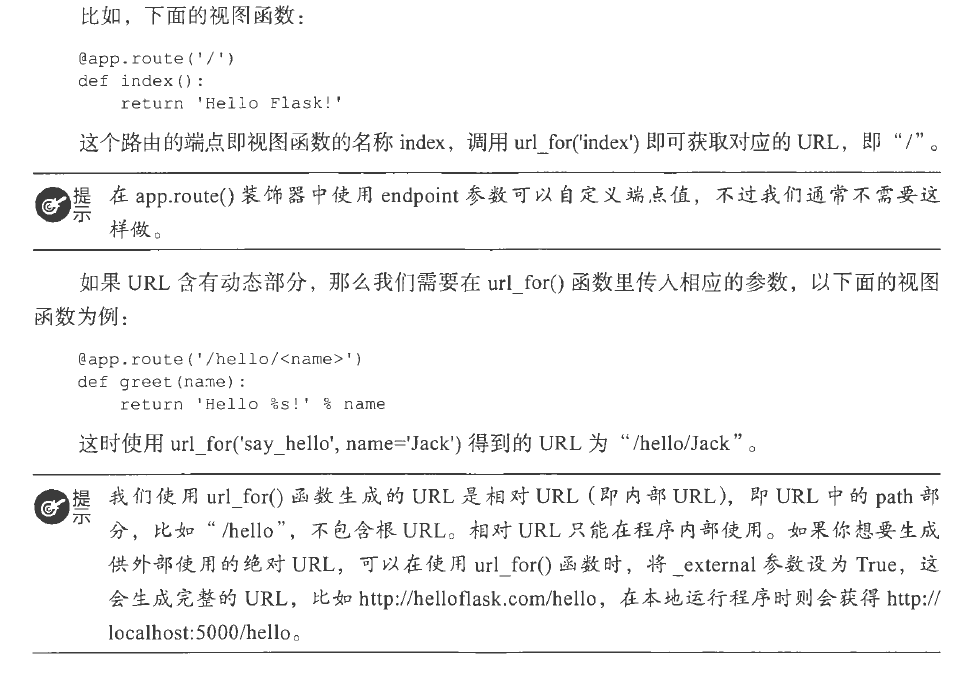
在Web 程序中，URL无处不在。如果程序中的URL 都是以硬编码的方式写出，那么将会

大大降低代码的易用性。

比如， 当你修改了某个路由的URL 规则，那么程序里对应的URL 都要一个一个进行修改。更好的解决办法是使用Flask 提供的**url\_for()**函数获取URL, 当路由中定义的URL 规则被修改时，这个函数总会返回正确的URL。

调用ur l\_for(）函数时，第一个参数为**端点(endpoint) 值**。在Fl ask 中，端点用来标记一个

视图函数以及对应的URL 规则。端点的默认值为视图函数的名称，至千为什么不直接使用视图函数名，而要引入端点这个概念，我们会在后面了解。



## Flask命令

除了Flask 内置的flask run 等命令，我们也可以自定义命令。在虚拟环境安装Flask 后，包

含许多内置命令的flask 脚本就可以使用了。在前面我们已经接触了很多flask 命令，比如运行服务器的flask run, 启动shell 的flask shell。

通过创建任意一个函数，并为其添加app.cli.command(） 装饰器，我们就可以注册一个flask

命令。代码清单1-4 创建了一个自定义的hello(）命令函数，在函数中我们仍然只是打印一行问候。