Flask-Assets

Release 0.12
Michael Elsdörfer

该文档由吴依杰根据Flask-Assets官方文档整理, 文档所有权归原作者所有

Flask- assets帮助您将webasset集成到Flask应用程序中

使用以下命令之一安装Flask-Assets

```
easy_install Flask-Assets
```

或者

```
pip install Flask-Assets
```

您可以通过创建一个环境实例来初始化应用程序, 并以所谓的包的形式向它注册您的Assets

```
from flask import Flask
from flask_assets import Environment, Bundle

app = Flask(__name__)
assets = Environment(app)

js = Bundle('jquery.js', 'base.js', 'widgets.js', filters='jsmin', output='gen/packed.js')
assets.register('js_all', js)
```

一个包由任意数量的源文件(它也可能包含其他嵌套的包)、一个输出目标和一个要应用的过滤器列表组成

所有路径都是相对于您的应用程序的`static`目录, 或`Flask Blueprint`的`static`目录

当然, 如果您愿意, 您也可以在一个外部配置文件中定义您的asserts, 并从那里读取它们.webasset为一些流行的格式(如YAML)提供了许多帮助类

与Flask的其他扩展相同, 通过初始化, Flask- asssets实例可以与多个应用程序一起使用, 通过`init_app()`调用, 而不是传递一个固定的应用对象

```
app = Flask(__name__)
assets = flask_assets.Environment()
assets.init_app(app)
```

使用包

现在你的assets正确定义, 你想合并和缩小他们, 并包括一个链接到压缩的结果在你的网页

```
{% assets "js_all" %}
<script type="text/javascript" src="{{ ASSET_URL }}"></script>
{% endassets %}
```

就这样,真的. 在第一次呈现模板时, `Flask-Assets`将自动合并和压缩包的源文件, 并在每次源文件更改时自动更新压缩文件. 如果将应用程序配置中的ASSETS_DEBUG设置为True，则每个源文件将分别输出.

Flask blueprints

如果你使用的是`Flask blueprints`, 你可以通过一个前缀来引用blueprint的静态文件, 就像Flask允许你引用blueprint的模板一样:

```
js = Bundle('app_level.js', 'blueprint/blueprint_level.js')
```

在上面的示例中, bundle将引用两个文件: `{APP_ROOT}/static/app_level.js`, `{BLUEPRINT_ROOT}/static/blueprint_level.js`

如果您以前单独使用过`webasset`库，那么您可能熟悉设置`directory`和`url`配置值的要求. 您将注意到, 这里不需要这样做, 因为使用的是Flask的静态文件夹支持. 但是, 请注意, 由于某些原因, 您可以根据需要设置自定义根目录或url. 但是, 在这种情况下,`Flask-Assets`的`blueprint`支持被禁用了, 也就是说, 使用前面描述的前缀引用不同蓝图中的静态文件不再可能. 所有路径都将考虑相对于您指定的目录和url.

0.7之前的模块也被支持;它们的工作方式完全相同

Templates only

如果你愿意, 你也可以不需要在代码中定义你的包, 简单地定义你模板中的一切:

```
{% assets filters="jsmin", output="gen/packed.js", "common/jquery.js", "site/base.js",
"site/widgets.js" %}
  <script type="text/javascript" src="{{ ASSET_URL }}"></script>
{% endassets %}
```

webasset支持两个配置选项, 既可以通过Environment实例设置, 也可以通过Flask配置设置. 以下两种说法是等价的

```
assets_env.debug = True
app.config['ASSETS_DEBUG'] = True
```

Babel Configuration

如果您使用Babel进行国际化, 那么您需要将扩展名`webasset .ext.jinja2. assetsextension`添加到Babel配置文件中:

```
[python: **.py]
[jinja2: **.html]
extensions = jinja2.ext.autoescape,jinja2.ext.with_,webassets.ext.jinja2.AssetsExtension
```

Flask-S3 Configuration

Flask-S3允许您从`Amazon S3 bucket`上传和提供静态文件. 它通过重写Flask `url_for`函数来实现这一点. 为了让`Flask-Assets`使用这个覆盖的`url_for`函数, 您需要让它知道您正在使用`Flask-S3`, 只需设置:

```
app.config['FLASK_ASSETS_USE_S3'] = True
```

Flask-CDN Configuration

`Flask-CDN`允许您从`CDN`(如Amazon Cloudfront)上传和提供静态文件, 而无需修改模板. 它通过覆盖Flask `url_for`函数来实现这一点. 为了让`Flask-Assets`使用这个覆盖的`url_for`函数, 您需要让它知道您正在使用`Flask-CDN`, 只需设置:

```
app.config['FLASK_ASSETS_USE_CDN'] = True
```

Version0.12 新加功能

Flask 0.11+提供了内置的CLI集成, 使用click库. 通过`setup.py`里的`entry_points`, 使用`flask.commands`可以自动安装asserts command.

安装Flask 0.11+后, 在你的shell中执行Flask命令的时候, 你会在输出中看到如下一行.

```
flask --help
...
Commands:
  assets    Web assets commands.
...
```

遗留的支持

如果你已经安装了`Flask-Script`, 那么一个可用命令将作为`flask_assets.ManageAssets`:

```
from flask_assets import ManageAssets

manager = Manager(app)
manager.add_command("assets", ManageAssets(assets_env))

```

您可以像上面那样在添加命令时显式地传递`assets_env`.或者,`ManageAssets`将从Flask导入`current_app`并使用`jinja_env`

该命令允许您做一些事情,比如从命令行重新构建包

在谷歌应用程序引擎使用

您可以在谷歌应用程序引擎中通过手动构建assets来使用`flask-assets`.`GAE`运行时不能创建文件(这是正常的磁盘资产功能所必需的), 但是您可以部署预构建的assets. 您可以使用文件更改侦听器在开发过程中动态地重新构建assets

API

webassets库与Flask的集成

`class flask_assets.Environment(app=None)`

此对象用于保存包和配置的集合, 如果它是用Flask应用程序的实例初始化的, 那么`webasset Jinja2`扩展会自动注册.

`config_storage_class` :FlaskConfigStorage的别名
`directory` :所有路径都相对的基本目录
`from_module` :来自Python模块的注册包
`from_yaml` :来自YAML配置文件的注册包
`resolver_class` :FlaskResolver的别名
`url` :所有静态url都相对于的基本url

`class flask_assets.Bundle(*contents, **options)`

bundle是webasset用来组织媒体文件组的单元, 用于应用哪些过滤器以及将它们存储在何处.

bundles可以任意嵌套.

关于绑定bundle和"environment"实例之间的连接的说明:绑定bundle需要它所属的环境.如果没有环境,它就缺乏关于如何行为的信息,并且无法知道相对路径的实际位置.但是,我不想做这个bundle.通过要求传递一个环境对象, 使得语法比以前更加复杂. 当使用嵌套bundle时, 这将是一个特别麻烦的问题. 而且, 嵌套的bundle永远不会显式地连接到环environment.而且, 相同的子bundle可以在多个父bundle中使用.

这就是为什么基本上Bundle类的每个方法都有一个env参数的原因, 这样一个父Bundle就可以为不知道它的子Bundle提供环境了.

`build(force=None, output=None, disable_cache=None)`

构建这个bundle, 意味着创建`output`属性提供的文件, 应用配置的过滤器等.

如果bundle是容器bundle,那么将构建多个文件.

除非给出`force`否则配置的`updater`将用于检查构建是否必要.

如果`output`是文件对象, 那么结果将被写入文件对象, 而不是写入文件系统.

返回值是`FileHunk`对象的列表, 每个对象对应一个构建的包.

depends

允许您定义一组附加的文件(支持glob语法), 在确定是否需要重新构建时将考虑这些文件

extra

附加到此绑定包的附加值的自定义用户dict. 这些将在模板标签中可用, 并可用于附加诸如CSS 'media'值之类的内容

get_version(ctx=None, refresh=False)

返回bundle的当前版本
如果版本没有缓存在内存中, 它将首先查看清单, 然后询问版本管理员
refresh会忽略内存中的一个值, 并重新查找版本

id()
这用于确定何时更改了bundle定义, 以便需要重新构建
因此, 应该根据实际影响最终构建结果的数据构建散列

is_container

如果这是一个容器bundle, 也就是说, 一个bundle仅作为许多子bundle的容器, 则返回true
它不能包含它自己的任何文件, 并且必须有一个空的输出属性

iterbuild(ctx)

迭代实际需要构建的bundle