Bottle: Python Web框架中文文档[¶](http://bottle.zzir.cn/index.html#bottle-python-web-framework)

Bottle是一个快速，简单，轻量级的 [Python](http://python.org/) [WSGI](http://www.wsgi.org/) Web框架。单一文件，只依赖 [Python标准库](http://docs.python.org/library/) 。

* **URL映射(Routing):** 将URL请求映射到Python函数，支持动态URL，且URL更简洁。
* **模板(Templates):** 快速且pythonic的 [内置模板引擎](http://bottle.zzir.cn/index.html#tutorial-templates) ，同时支持 [mako](http://www.makotemplates.org/) , [jinja2](http://jinja.pocoo.org/) 和 [cheetah](http://www.cheetahtemplate.org/) 等模板。
* **基础功能(Utilities):** 方便地访问表单数据，上传文件，使用cookie，查看HTTP元数据。
* **服务器(Server):** 内置了开发服务器，且支持 [paste](http://pythonpaste.org/), [fapws3](https://github.com/william-os4y/fapws3), [bjoern](https://github.com/jonashaag/bjoern), [gae](https://developers.google.com/appengine/), [cherrypy](http://www.cherrypy.org/) 等符合 [WSGI](http://www.wsgi.org/) 标准的 HTTP 服务器。

**示例： “Hello World”**

**from** **bottle** **import** route, run, template

**@route**('/hello/<name>')

**def** index(name):

**return** template('<b>Hello {{name}}</b>!', name=name)

run(host='localhost', port=8080)

将其保存为py文件并执行，用浏览器访问 <http://localhost:8080/hello/world> 即可看到效果。就这么简单！

**下载和安装**

通过 pip install bottle 或 easy\_install -U bottle 安装最新稳定版，再或者下载 [bottle.py](https://github.com/bottlepy/bottle/raw/master/bottle.py) (开发版) 到你的项目目录。 Bottle 除了 Python 标准库外无任何依赖库[[1]](http://bottle.zzir.cn/index.html#id3)。 且支持 **Python 2.6+** 和 **Python 3.2+**。

用户指南[¶](http://bottle.zzir.cn/index.html#user-s-guide)

如果你有将Bottle用于Web开发的打算，请继续看下去。如果这份文档没有解决你遇到的问题，尽管在 [邮件列表](mailto:bottlepy%40googlegroups.com) 中吼出来吧（译者注：用英文哦）。

教程[¶](http://bottle.zzir.cn/index.html#tutorial)

这份教程将向你介绍Bottle的开发理念和功能特性。既介绍Bottle的基本用法，也包含了进阶用法。你可以从头到尾通读一遍，也可当做开发时的参考。你也许对自动生成的 [API参考](http://bottle.zzir.cn/index.html#document-api) 感兴趣。它包含了更多的细节，但解释没有这份教程详细。在 [秘诀](http://bottle.zzir.cn/index.html#document-recipes) 或 [常见问题](http://bottle.zzir.cn/index.html#document-faq) 可找到常见问题的解决办法。如果需要任何帮助，可加入我们的 [邮件列表](mailto:bottlepy%40googlegroups.com) 或在 [IRC频道](http://webchat.freenode.net/?channels=bottlepy) 和我们交流。

安装[¶](http://bottle.zzir.cn/index.html#installation)

Bottle不依赖其他库，你需要做的仅是下载 [bottle.py](http://bottle.zzir.cn/bottle.py) (开发版)到你的项目文件夹，然后开始写代码。

$ wget http://bottlepy.org/bottle.py

在终端运行以上命令，即可下载到Bottle的最新开发版，包含了所有新功能特性。如果更需要稳定性，你应该坚持使用Bottle的稳定版本。可在 [PyPI](http://pypi.python.org/pypi/bottle) 下载稳定版本，然后通过 **pip** (推荐), **easy\_install** 或你的包管理软件安装。

$ sudo pip install bottle *# 推荐*

$ sudo easy\_install bottle *# 若无pip，尝试这个*

$ sudo apt-get install python-bottle *# 适用于 debian, ubuntu, ...*

总之，你需要 Python 2.6 或以上 (包括 3.2+) 版本来运行Bottle。 如果你没有权限或不想安装全局性的Bottle，可以尝试使用 [virtualenv](http://pypi.python.org/pypi/virtualenv)：

$ virtualenv develop *# 创建虚拟环境*

$ source develop/bin/activate *# 激活虚拟环境*

(develop)$ pip install -U bottle *# 在虚拟环境中安装 Bottle*

如果还未安装virtualenv:

$ wget https://raw.github.com/pypa/virtualenv/master/virtualenv.py

$ python virtualenv.py develop *# 创建一个虚拟环境*

$ source develop/bin/activate *# 使用虚拟环境里的 Python 解析器*

(develop)$ pip install -U bottle *# 在虚拟环境中安装 Bottle*

开始： “Hello World”[¶](http://bottle.zzir.cn/index.html#quickstart-hello-world)

到目前为止，我假设你已经 [安装](http://bottle.zzir.cn/index.html#installation) 好了bottle或已将bottle.py拷贝到你的项目文件夹。接下来我们就可以写一个非常简单的”Hello World”了:

**from** **bottle** **import** route, run

**@route**('/hello')

**def** hello():

**return** "Hello World!"

run(host='localhost', port=8080, debug=True)

就这么简单！保存为py文件并执行，用浏览器访问 <http://localhost:8080/hello> 就可以看到”Hello World!”。它的执行流程大致如下:

[**route()**](http://bottle.zzir.cn/index.html#bottle.route) 函数将一段代码绑定到一个 URL，在这个例子中，我们将 hello() 函数绑定给了 /hello。 我们称之为 *route* (也是该修饰器的函数名)，这也是 Bottle 框架最重要的开发理念。在浏览器请求一个 URL 的时候，框架自动调用与之相应的函数，接着将函数的返回值发送给浏览器。就这么简单！

最后一行调用的 [**run()**](http://bottle.zzir.cn/index.html#bottle.run) 函数启动了内置的开发服务器。它监听 *localhost* 的8080端口并响应请求， Control-c 可将其关闭。到目前为止，这个内置的开发服务器已经足够用于日常的开发测试了。它根本不需要安装，就可以让你的应用跑起来。在教程的后面，你将学会如何让你的应用跑在其他服务器上面（译者注：内置服务器不能满足生产环境的要求）

[调试模式](http://bottle.zzir.cn/index.html#tutorial-debugging) 在早期开发的时候非常有用，但请务必记得，在生产环境中将其关闭。

毫无疑问，这是一个十分简单的例子，但它展示了 Bottle 开发应用的基本理念。接下来你将了解到其它开发方式。

*默认应用*[¶](http://bottle.zzir.cn/index.html#the-default-application)

基于简单性考虑，这份教程中的大部分例子都使用一个模块层面的 [**route()**](http://bottle.zzir.cn/index.html#bottle.route) 修饰器函数来定义route。这样的话，所有route都添加到了一个全局的“默认应用”里面，即是在第一次调用 [**route()**](http://bottle.zzir.cn/index.html#bottle.route) 函数时，创建的一个 [**Bottle**](http://bottle.zzir.cn/index.html#bottle.Bottle) 类的实例。其他几个模块层面的修饰器函数都与这个“默认应用”有关，如果你偏向于面向对象的做法且不介意多打点字，你可以创建一个独立的应用对象，这样就可避免使用全局范围的“默认应用”。

**from** **bottle** **import** Bottle, run

app = Bottle()

**@app.route**('/hello')

**def** hello():

**return** "Hello World!"

run(app, host='localhost', port=8080)

接下来的 [默认应用](http://bottle.zzir.cn/index.html#default-app) 章节中将更详细地介绍这种做法。现在，你只需知道不止有一种选择就好了。

URL映射[¶](http://bottle.zzir.cn/index.html#request-routing)

在上一章中，我们实现了一个十分简单的web应用，只有一个URL映射(route)。让我们再来看一下“Hello World”中与routing有关的部分:

**@route**('/hello')

**def** hello():

**return** "Hello World!"

[**route()**](http://bottle.zzir.cn/index.html#bottle.route) 函数将一个 URL 路径与一个回调函数关联了起来，然后在 [default application](http://bottle.zzir.cn/index.html#tutorial-default)中添加了一个 URL 映射 (route)。但只有一个 route 的应用未免太无聊了，让我们试着再添加几个 route 吧。(不要忘了 from bottle import template)：

**@route**('/')

**@route**('/hello/<name>')

**def** greet(name='Stranger'):

**return** template('Hello {{name}}, how are you?', name=name)

这个例子说明了两件事情，一个回调函数可绑定多个route，你也可以在URL中添加通配符，然后在回调函数中使用它们。

动态URL映射[¶](http://bottle.zzir.cn/index.html#dynamic-routes)

包含通配符的route，我们称之为动态route(与之对应的是静态route)，它能匹配多个URL地址。一个通配符包含在一对尖括号里面(像这样 <name> )，通配符之间用”/”分隔开来。如果我们将URL定义为 /hello/<name> 这样，那么它就能匹配 /hello/alice 和 /hello/bob 这样的浏览器请求，但不能匹配 /hello , /hello/ 和 /hello/mr/smith 。

URL中的通配符都会当作参数传给回调函数，直接在回调函数中使用。这样可以漂亮地实现RESTful形式的URL。例子如下:

**@route**('/wiki/<pagename>') *# 匹配 /wiki/Learning\_Python*

**def** show\_wiki\_page(pagename):

...

**@route**('/<action>/<user>') *# 匹配(matches) /follow/defnull*

**def** user\_api(action, user):

...

过滤器 (Filter) 可被用来定义特殊类型的通配符，在传通配符给回调函数之前，先自动转换通配符类型。包含过滤器的通配符定义一般像 <name:filter> 或 <name:filter:config>。 config 部分是可选的，其语法由你使用的过滤器决定。

已实现下面几种形式的过滤器，后续可能会继续添加:

* **:int** 匹配一个数字，自动将其转换为int类型。
* **:float** 与:int类似，用于浮点数。
* **:path** 匹配一个路径(包含”/”)
* **:re** 匹配config部分的一个正则表达式，不更改被匹配到的值

让我们来看看具体的使用例子:

**@route**('/object/<id:int>')

**def** callback(id):

**assert** isinstance(id, int)

**@route**('/show/<name:re:[a-z]+>')

**def** callback(name):

**assert** name.isalpha()

**@route**('/static/<path:path>')

**def** callback(path):

**return** static\_file(path, ...)

你可以添加自己的过滤器。详见 [URL映射](http://bottle.zzir.cn/index.html#document-routing)。

HTTP请求方法[¶](http://bottle.zzir.cn/index.html#http-request-methods)

HTTP 协议定义了多个 [请求方法](http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html) (有时被称为 “verbs”) 来满足不同的需求。 Route 默认使用 GET 方法，通过匹配响应。处理其它方法像 POST, PUT, DELETE , PATCH等， 通过给 [**route()**](http://bottle.zzir.cn/index.html#bottle.route)函数指定 method 参数。或使用以下5种装饰器： [**get()**](http://bottle.zzir.cn/index.html#bottle.get), [**post()**](http://bottle.zzir.cn/index.html#bottle.post), [**put()**](http://bottle.zzir.cn/index.html#bottle.put), [**delete()**](http://bottle.zzir.cn/index.html#bottle.delete) 或 [**patch()**](http://bottle.zzir.cn/index.html#bottle.patch)。

POST方法一般用于HTML表单的提交。下面是一个使用POST来实现用户登录的例子:

**from** **bottle** **import** get, post, request *# or route*

**@get**('/login') *# or @route('/login')*

**def** login():

**return** '''

<form action="/login" method="post">

Username: <input name="username" type="text" />

Password: <input name="password" type="password" />

<input value="Login" type="submit" />

</form>

'''

**@post**('/login') *# or @route('/login', method='POST')*

**def** do\_login():

username = request.forms.get('username')

password = request.forms.get('password')

**if** check\_login(username, password):

**return** "<p>Your login information was correct.</p>"

**else**:

**return** "<p>Login failed.</p>"

在这个例子中， /login 绑定了两个回调函数，一个回调函数响应GET请求，一个回调函数响应POST请求。如果浏览器使用GET请求访问 /login ，则调用login\_form()函数来返回登录页面，浏览器使用POST方法提交表单后，调用login\_submit()函数来检查用户有效性，并返回登录结果。接下来的 [请求数据 (Request Data)](http://bottle.zzir.cn/index.html#tutorial-request) 章节中，会详细介绍 **Request.forms** 的用法。

**特殊请求方法: HEAD 和 ANY**

HEAD方法类似于GET方法，但服务器不会返回HTTP响应正文，一般用于获取HTTP原数据而不用下载整个页面。Bottle像处理GET请求那样处理HEAD请求，但是会自动去掉HTTP响应正文。你无需亲自处理HEAD请求。

另外，非标准的ANY方法做为一个低优先级的fallback：在没有其它route的时候，监听ANY方法的route会匹配所有请求，而不管请求的方法是什么。这对于用做代理的route很有用，可将所有请求都重定向给子应用。

总而言之：HEAD请求被响应GET请求的route来处理，响应ANY请求的route处理所有请求，但仅限于没有其它route来匹配原先的请求的情况。就这么简单。

静态文件映射[¶](http://bottle.zzir.cn/index.html#routing-static-files)

Bottle不会处理像图片或CSS文件的静态文件请求。你需要给静态文件提供一个route，一个回调函数(用于查找和控制静态文件的访问)。

**from** **bottle** **import** static\_file

**@route**('/static/<filename>')

**def** server\_static(filename):

**return** static\_file(filename, root='/path/to/your/static/files')

**static\_file()** 函数用于响应静态文件的请求。 (详见 [静态文件](http://bottle.zzir.cn/index.html#tutorial-static-files) )这个例子只能响应在 /path/to/your/static/files 目录下的文件请求，因为 <filename> 这样的通配符定义不能匹配一个路径(路径中包含”/”)。 为了响应子目录下的文件请求，我们需要更改 *path* 过滤器的定义:

**@route**('/static/<filepath:path>')

**def** server\_static(filepath):

**return** static\_file(filepath, root='/path/to/your/static/files')

使用 root='./static/files' 这样的相对路径的时候，请注意当前工作目录 (./) 不一定是项目文件夹。

错误页面[¶](http://bottle.zzir.cn/index.html#error-pages)

如果出错了，Bottle会显示一个默认的错误页面，提供足够的debug信息。你也可以使用 [**error()**](http://bottle.zzir.cn/index.html#bottle.error) 函数来自定义你的错误页面:

**from** **bottle** **import** error

**@error**(404)

**def** error404(error):

**return** 'Nothing here, sorry'

从现在开始，在遇到404错误的时候，将会返回你在上面自定义的页面。传给error404函数的唯一参数，是一个 [**HTTPError**](http://bottle.zzir.cn/index.html#bottle.HTTPError) 对象的实例。除此之外，这个回调函数与我们用来响应普通请求的回调函数没有任何不同。你可以从 [**request**](http://bottle.zzir.cn/index.html#bottle.request) 中读取数据， 往 [**response**](http://bottle.zzir.cn/index.html#bottle.response) 中写入数据和返回所有支持的数据类型，除了 [**HTTPError**](http://bottle.zzir.cn/index.html#bottle.HTTPError) 的实例。

只有在你的应用返回或raise一个 [**HTTPError**](http://bottle.zzir.cn/index.html#bottle.HTTPError) 异常的时候(就像 **abort()** 函数那样)，处理Error的函数才会被调用。更改 **Request.status** 或返回 [**HTTPResponse**](http://bottle.zzir.cn/index.html#bottle.HTTPResponse) 不会触发错误处理函数。

生成内容[¶](http://bottle.zzir.cn/index.html#generating-content)

在纯WSGI环境里，你的应用能返回的内容类型相当有限。应用必须返回一个iterable的字节型字符串。你可以返回一个字符串(因为字符串是iterable的)，但这会导致服务器按字符来传输你的内容。Unicode字符串根本不允许。这不是很实用。

Bottle支持返回更多的内容类型，更具弹性。它甚至能在合适的情况下，在HTTP头中添加 *Content-Length* 字段和自动转换unicode编码。下面列出了所有你能返回的内容类型，以及框架处理方式的一个简述。

Dictionaries

上面已经提及，Python中的字典类型(或其子类)会被自动转换为JSON字符串。返回给浏览器的时候，HTTP头的 Content-Type 字段被自动设置为 application/json。可十分简单地实现基于JSON的API。Bottle同时支持json之外的数据类型，详见 tutorial-output-filter 。

Empty Strings, False, None or other non-true values:

输出为空， Content-Length 设为0。

Unicode的问题

Unicode字符串 (or iterables yielding unicode strings) 被自动转码， Content-Type 被默认设置为utf8，接着视之为普通字符串(见下文)。

Byte strings

Bottle将字符串当作一个整体来返回(而不是按字符来遍历)，并根据字符串长度添加 Content-Length 字段。包含字节型字符串的列表先被合并。其它iterable的字节型字符串不会被合并，因为它们也许太大来，耗内存。在这种情况下， Content-Length 字段不会被设置。

Instances of [**HTTPError**](http://bottle.zzir.cn/index.html#bottle.HTTPError) or [**HTTPResponse**](http://bottle.zzir.cn/index.html#bottle.HTTPResponse)

返回它们和直接raise出来有一样的效果。对于 [**HTTPError**](http://bottle.zzir.cn/index.html#bottle.HTTPError) 来说，会调用错误处理程序。详见 [错误页面](http://bottle.zzir.cn/index.html#tutorial-errorhandling) 。

File objects

任何有 .read() 方法的对象都被当成一个file-like对象来对待，会被传给 WSGI Server 框架定义的 wsgi.file\_wrapper callable对象来处理。一些WSGI Server实现会利用优化过的系统调用(sendfile)来更有效地传输文件，另外就是分块遍历。可选的HTTP头，例如 Content-Length 和 Content-Type 不会被自动设置。尽可能使用 **send\_file()** 。详见 [静态文件](http://bottle.zzir.cn/index.html#tutorial-static-files) 。

Iterables and generators

你可以在回调函数中使用 yield 语句，或返回一个iterable的对象，只要该对象返回的是字节型字符串，unicode字符串， [**HTTPError**](http://bottle.zzir.cn/index.html#bottle.HTTPError) 或 [**HTTPResponse**](http://bottle.zzir.cn/index.html#bottle.HTTPResponse) 实例。不支持嵌套iterable对象，不好意思。注意，在iterable对象返回第一个非空值的时候，就会把HTTP状态码和HTTP头发送给浏览器。稍后再更改它们就起不到什么作用了。

以上列表的顺序非常重要。在你返回一个 **str** 类的子类的时候，即使它有 .read() 方法，它依然会被当成一个字符串对待，而不是文件，因为字符串先被处理。

**改变默认编码**

Bottle使用 Content-Type 的 *charset* 参数来决定编码unicode字符串的方式。默认的 Content-Type 是 text/html;charset=UTF8 ，可在 **Response.content\_type** 属性中修改，或直接设置 **Response.charset** 的值。关于 [**Response**](http://bottle.zzir.cn/index.html#bottle.Response) 对象的介绍，详见 [Response 对象](http://bottle.zzir.cn/index.html#tutorial-response) 。

**from** **bottle** **import** response

**@route**('/iso')

**def** get\_iso():

response.charset = 'ISO-8859-15'

**return** u'This will be sent with ISO-8859-15 encoding.'

**@route**('/latin9')

**def** get\_latin():

response.content\_type = 'text/html; charset=latin9'

**return** u'ISO-8859-15 is also known as latin9.'

在极少情况下，Python中定义的编码名字和HTTP标准中的定义不一样。这样，你就必须同时修改 **Response.content\_type** (发送给客户端的)和设置 **Response.charset** 属性 (用于编码unicode)。

静态文件[¶](http://bottle.zzir.cn/index.html#static-files)

你可直接返回文件对象。但我们更建议你使用 **static\_file()** 来提供静态文件服务。它会自动猜测文件的 mime-type，添加 Last-Modified 头，将文件路径限制在一个 root 文件夹下面来保证安全，且返回合适的 HTTP 状态码 (权限不足导致的403错误，文件不存在导致的404错误)。它甚至支持 If-Modified-Since 头，如果文件未被修改，则直接返回 304 Not Modified 。你可指定 MIME 类型来避免其自动猜测。

**from** **bottle** **import** static\_file

**@route**('/images/<filename:re:.\*\.png>')

**def** send\_image(filename):

**return** static\_file(filename, root='/path/to/image/files', mimetype='image/png')

**@route**('/static/<filename:path>')

**def** send\_static(filename):

**return** static\_file(filename, root='/path/to/static/files')

如果确实需要，你可将 **static\_file()** 的返回值当作异常raise出来。

**强制下载**

大多数浏览器在知道MIME类型的时候，会尝试直接调用相关程序来打开文件(例如PDF文件)。如果你不想这样，你可强制浏览器只是下载该文件，甚至提供文件名。:

**@route**('/download/<filename:path>')

**def** download(filename):

**return** static\_file(filename, root='/path/to/static/files', download=filename)

如果 download 参数的值为 True ，会使用原始的文件名。

HTTP错误和重定向[¶](http://bottle.zzir.cn/index.html#http-errors-and-redirects)

**abort()** 函数是生成HTTP错误页面的一个捷径。

**from** **bottle** **import** route, abort

**@route**('/restricted')

**def** restricted():

abort(401, "Sorry, access denied.")

为了将用户访问重定向到其他URL，你在 Location 中设置新的URL，接着返回一个 303 See Other 。 **redirect()** 函数可以帮你做这件事情。

**from** **bottle** **import** redirect

**@route**('/wrong/url')

**def** wrong():

redirect("/right/url")

你可以在第二个参数中提供另外的HTTP状态码。

注解

这两个函数都会抛出 [**HTTPError**](http://bottle.zzir.cn/index.html#bottle.HTTPError) 异常，终止回调函数的执行。

**其他异常**

除了 [**HTTPResponse**](http://bottle.zzir.cn/index.html#bottle.HTTPResponse) 或 [**HTTPError**](http://bottle.zzir.cn/index.html#bottle.HTTPError) 以外的其他异常，都会导致500错误，所以不会造成WSGI服务器崩溃。你将 bottle.app().catchall 的值设为 False 来关闭这种行为，以便在你的中间件中处理异常。

[**Response**](http://bottle.zzir.cn/index.html#bottle.Response) 对象[¶](http://bottle.zzir.cn/index.html#the-response-object)

诸如HTTP状态码，HTTP响应头，用户cookie等元数据都保存在一个名字为 [**response**](http://bottle.zzir.cn/index.html#bottle.response) 的对象里面，接着被传输给浏览器。你可直接操作这些元数据或使用一些更方便的函数。在API章节可查到所有相关API(详见 [**Response**](http://bottle.zzir.cn/index.html#bottle.Response) )，这里主要介绍一些常用方法。

**状态码**

[HTTP状态码](http://bottle.zzir.cn/http_code) 控制着浏览器的行为，默认为 200 OK 。多数情况下，你不必手动修改 **Response.status** 的值，可使用 **abort()** 函数或return一个 [**HTTPResponse**](http://bottle.zzir.cn/index.html#bottle.HTTPResponse) 实例(带有合适的状态码)。虽然所有整数都可当作状态码返回，但浏览器不知道如何处理 [HTTP标准](http://bottle.zzir.cn/http_code) 中定义的那些状态码之外的数字，你也破坏了大家约定的标准。

**响应头**

Cache-Control 和 Location 之类的响应头通过 **Response.set\_header()** 来定义。这个方法接受两个参数，一个是响应头的名字，一个是它的值，名字是大小写敏感的。

**@route**('/wiki/<page>')

**def** wiki(page):

response.set\_header('Content-Language', 'en')

...

大多数的响应头是唯一的，meaning that only one header per name is send to the client。一些特殊的响应头在一次response中允许出现多次。使用 **Response.add\_header()** 来添加一个额外的响应头，而不是 **Response.set\_header()**

response.set\_header('Set-Cookie', 'name=value')

response.add\_header('Set-Cookie', 'name2=value2')

请注意，这只是一个例子。如果你想使用cookie，详见 [ahead](http://bottle.zzir.cn/index.html#tutorial-cookies) 。

Cookies[¶](http://bottle.zzir.cn/index.html#cookies)

Cookie是储存在浏览器配置文件里面的一小段文本。你可通过 **Request.get\_cookie()** 来访问已存在的Cookie，或通过 **Response.set\_cookie()** 来设置新的Cookie。

**@route**('/hello')

**def** hello\_again():

**if** request.get\_cookie("visited"):

**return** "Welcome back! Nice to see you again"

**else**:

response.set\_cookie("visited", "yes")

**return** "Hello there! Nice to meet you"

**Response.set\_cookie()** 方法接受一系列额外的参数，来控制Cookie的生命周期及行为。一些常用的设置如下:

* **max\_age:** 最大有效时间，以秒为单位 (默认: None)
* **expires:** 一个datetime对象或一个UNIX timestamp (默认: None)
* **domain:** 可访问该Cookie的域名 (默认: 当前域名)
* **path:** 限制cookie的访问路径 (默认: /)
* **secure:** 只允许在HTTPS链接中访问cookie (默认: off)
* **httponly:** 防止客户端的javascript读取cookie (默认: off, 要求python 2.6或以上版本)

如果 *expires* 和 *max\_age* 两个值都没设置，cookie会在当前的浏览器session失效或浏览器窗口关闭后失效。在使用cookie的时候，应该注意一下几个陷阱。

* 在大多数浏览器中，cookie的最大容量为4KB。
* 一些用户将浏览器设置为不接受任何cookie。大多数搜索引擎也忽略cookie。确保你的应用在无cookie的时候也能工作。
* cookie被储存在客户端，也没被加密。你在cookie中储存的任何数据，用户都可以读取。更坏的情况下，cookie会被攻击者通过 [XSS](http://en.wikipedia.org/wiki/HTTP_cookie#Cookie_theft_and_session_hijacking) 偷走，一些已知病毒也会读取浏览器的cookie。既然如此，就不要在cookie中储存任何敏感信息。
* cookie可以被伪造，不要信任cookie！

**Cookie签名**

上面提到，cookie容易被客户端伪造。Bottle可通过加密cookie来防止此类攻击。你只需在读取和设置cookie的时候，通过 *secret* 参数来提供一个密钥。如果cookie未签名或密钥不匹配， **Request.get\_cookie()** 方法返回 None

**@route**('/login')

**def** do\_login():

username = request.forms.get('username')

password = request.forms.get('password')

**if** check\_login(username, password):

response.set\_cookie("account", username, secret='some-secret-key')

**return** template("<p>Welcome {{name}}! You are now logged in.</p>", name=username)

**else**:

**return** "<p>Login failed.</p>"

**@route**('/restricted')

**def** restricted\_area():

username = request.get\_cookie("account", secret='some-secret-key')

**if** username:

**return** template("Hello {{name}}. Welcome back.", name=username)

**else**:

**return** "You are not logged in. Access denied."

例外，Bottle自动序列化储存在签名cookie里面的数据。你可在cookie中储存任何可序列化的对象(不仅仅是字符串)，只要对象大小不超过4KB。

警告

签名cookie在客户端不加密(译者注：即在客户端没有经过二次加密)，也没有写保护(客户端可使用之前的cookie)。给cookie签名的主要意义在于在cookie中存储序列化对象和防止伪造cookie，依然不要在cookie中存储敏感信息。

请求数据 (Request Data)[¶](http://bottle.zzir.cn/index.html#request-data)

可通过全局的 [**request**](http://bottle.zzir.cn/index.html#bottle.request) 对象来访问Cookies，HTTP头，HTML的 <form> 字段，以及其它的请求数据。这个特殊的对象总是指向 *当前* 的请求，即使在同时处理多个客户端连接的多线程情况下。

**from** **bottle** **import** request, route, template

**@route**('/hello')

**def** hello():

name = request.cookies.username **or** 'Guest'

**return** template('Hello {{name}}', name=name)

[**request**](http://bottle.zzir.cn/index.html#bottle.request) 对象继承自 [**BaseRequest**](http://bottle.zzir.cn/index.html#bottle.BaseRequest) ，提供了丰富的API来访问数据。虽然我们只介绍最常用的特性，也足够入门了。

介绍 [**FormsDict**](http://bottle.zzir.cn/index.html#bottle.FormsDict)[¶](http://bottle.zzir.cn/index.html#introducing-formsdict)

Bottle使用了一个特殊的字典来储存表单数据和cookies。 [**FormsDict**](http://bottle.zzir.cn/index.html#bottle.FormsDict) 表现得像一个普通的字典，但提供了更方便的额外功能。

**属性访问** ：字典中所有的值都可以当做属性来访问。这些虚拟的属性返回unicode字符串。在字典中缺少对应的值，或unicode解码失败的情况下，属性返回的字符串为空。

name = request.cookies.name

*# is a shortcut for:*

name = request.cookies.getunicode('name') *# encoding='utf-8' (default)*

*# which basically does this:*

**try**:

name = request.cookies.get('name', '').decode('utf-8')

**except** UnicodeError:

name = u''

**一个key对应多个value：** [**FormsDict**](http://bottle.zzir.cn/index.html#bottle.FormsDict) 是 **MutilDict** 的子类，一个key可存储多个value。标准的字典访问方法只返回一个值，但 [**getall()**](http://bottle.zzir.cn/index.html#bottle.MultiDict.getall) 方法会返回一个包含了所有value的一个list（也许为空）。

**for** choice **in** request.forms.getall('multiple\_choice'):

do\_something(choice)

**WTForms支持：** 一些第三方库（例如 [WTForms](http://wtforms.simplecodes.com/) ）希望输入中的所有字典都是unicode的。 [**FormsDict.decode()**](http://bottle.zzir.cn/index.html#bottle.FormsDict.decode) 帮你做了这件事情。它将所有value重新编码，并返回原字典的一个拷贝，同时保留所有特性，例如一个key对应多个value。

注解

在 **Python2** 中，所有的key和value都是byte-string。如果你需要unicode，可使用 [**FormsDict.getunicode()**](http://bottle.zzir.cn/index.html#bottle.FormsDict.getunicode) 方法或像访问属性那样访问。这两种方法都试着将字符串转码(默认: utf8)，如果失败，将返回一个空字符串。无需捕获 **UnicodeError** 异常。

**>>>** request.query['city']

'G\xc3\xb6ttingen' # A utf8 byte string

**>>>** request.query.city

u'Göttingen' # The same string as unicode

在 **Python3** 中，所有的字符串都是unicode。但HTTP是基于字节的协议，在byte-string被传给应用之前，服务器必须将其转码。安全起见，WSGI协议建议使用ISO-8859-1 (即是latin1)，一个可反转的单字节编码，可被转换为其他编码。Bottle通过 [**FormsDict.getunicode()**](http://bottle.zzir.cn/index.html#bottle.FormsDict.getunicode) 和属性访问实现了转码，但不支持字典形式的访问。通过字典形式的访问，将直接返回服务器返回的字符串，未经处理，这或许不是你想要的。

**>>>** request.query['city']

'GÃ¶ttingen' # An utf8 string provisionally decoded as ISO-8859-1 by the server

**>>>** request.query.city

'Göttingen' # The same string correctly re-encoded as utf8 by bottle

如果你整个字典包含正确编码后的值(e.g. for WTForms)，可通过 [**FormsDict.decode()**](http://bottle.zzir.cn/index.html#bottle.FormsDict.decode) 方法来获取一个转码后的拷贝(译者注：一个新的实例)。

Cookies[¶](http://bottle.zzir.cn/index.html#id3)

Cookie是客户端浏览器存储的一些文本数据，在每次请求的时候发送回给服务器。Cookie被用于在多次请求间保留状态信息（HTTP本身是无状态的），但不应该用于保存安全相关信息。因为客户端很容易伪造Cookie。

可通过 [**BaseRequest.cookies**](http://bottle.zzir.cn/index.html#bottle.BaseRequest.cookies) (一个 [**FormsDict**](http://bottle.zzir.cn/index.html#bottle.FormsDict)) 来访问所有客户端发来的Cookie。下面的是一个基于Cookie的访问计数。

**from** **bottle** **import** route, request, response

**@route**('/counter')

**def** counter():

count = int( request.cookies.get('counter', '0') )

count += 1

response.set\_cookie('counter', str(count))

**return** 'You visited this page *%d* times' % count

[**BaseRequest.get\_cookie()**](http://bottle.zzir.cn/index.html#bottle.BaseRequest.get_cookie) 是访问cookie的另一种方法。它支持解析 [signed cookies](http://bottle.zzir.cn/index.html#tutorial-signed-cookies) 。

HTTP头[¶](http://bottle.zzir.cn/index.html#http-headers)

所有客户端发送过来的HTTP头(例如 Referer, Agent 和 Accept-Language)存储在一个 [**WSGIHeaderDict**](http://bottle.zzir.cn/index.html#bottle.WSGIHeaderDict) 中，可通过 [**BaseRequest.headers**](http://bottle.zzir.cn/index.html#bottle.BaseRequest.headers) 访问。 [**WSGIHeaderDict**](http://bottle.zzir.cn/index.html#bottle.WSGIHeaderDict) 基本上是一个字典，其key大小写敏感。

**from** **bottle** **import** route, request

**@route**('/is\_ajax')

**def** is\_ajax():

**if** request.headers.get('X-Requested-With') == 'XMLHttpRequest':

**return** 'This is an AJAX request'

**else**:

**return** 'This is a normal request'

查询变量[¶](http://bottle.zzir.cn/index.html#query-variables)

查询字符串(例如 /forum?id=1&page=5 )一般用于向服务器传输键值对。你可通过 [**BaseRequest.query**](http://bottle.zzir.cn/index.html#bottle.BaseRequest.query) ( [**FormsDict**](http://bottle.zzir.cn/index.html#bottle.FormsDict) 类的实例) 来访问，和通过 [**BaseRequest.query\_string**](http://bottle.zzir.cn/index.html#bottle.BaseRequest.query_string) 来获取整个字符串。

**from** **bottle** **import** route, request, response, template

**@route**('/forum')

**def** display\_forum():

forum\_id = request.query.id

page = request.query.page **or** '1'

**return** template('Forum ID: {{id}} (page {{page}})', id=forum\_id, page=page)

处理HTML的 *<form>* 标签[¶](http://bottle.zzir.cn/index.html#html-form-handling)

让我们从头开始。在HTML中，一个典型的 <form> 标签看起来是这样的。

<**form** action="/login" method="post">

Username: <**input** name="username" type="text" />

Password: <**input** name="password" type="password" />

<**input** value="Login" type="submit" />

</**form**>

action 属性指定了用于接收表单数据的URL， method 定义了使用的HTTP方法（ GET 或 POST ）。如果使用GET方法，表单中的数据会附加到URL后面，可通过 [**BaseRequest.query**](http://bottle.zzir.cn/index.html#bottle.BaseRequest.query) 来访问。这被认为是不安全的，且有其它限制。所以这里我们使用POST方法。如果有疑惑，就使用 POST 吧。

通过POST方法传输的表单字段，作为一个 [**FormsDict**](http://bottle.zzir.cn/index.html#bottle.FormsDict) 存储在 [**BaseRequest.forms**](http://bottle.zzir.cn/index.html#bottle.BaseRequest.forms) 中。服务器端的代码看起来是这样的。

**from** **bottle** **import** route, request

**@route**('/login')

**def** login():

**return** '''

<form action="/login" method="post">

Username: <input name="username" type="text" />

Password: <input name="password" type="password" />

<input value="Login" type="submit" />

</form>

'''

**@route**('/login', method='POST')

**def** do\_login():

username = request.forms.get('username')

password = request.forms.get('password')

**if** check\_login(username, password):

**return** "<p>Your login information was correct.</p>"

**else**:

**return** "<p>Login failed.</p>"

有其它一些属性也可以用来访问表单数据。为了方便，一些属性包含了多个来源的数据。下面的表格可给你一个直观的印象。

| **属性** | **GET Form fields** | **POST表单数据** | **File Uploads** |
| --- | --- | --- | --- |
| [**BaseRequest.query**](http://bottle.zzir.cn/index.html#bottle.BaseRequest.query) | yes | no | no |
| [**BaseRequest.forms**](http://bottle.zzir.cn/index.html#bottle.BaseRequest.forms) | no | yes | no |
| [**BaseRequest.files**](http://bottle.zzir.cn/index.html#bottle.BaseRequest.files) | no | no | yes |
| [**BaseRequest.params**](http://bottle.zzir.cn/index.html#bottle.BaseRequest.params) | yes | yes | no |
| [**BaseRequest.GET**](http://bottle.zzir.cn/index.html#bottle.BaseRequest.GET) | yes | no | no |
| [**BaseRequest.POST**](http://bottle.zzir.cn/index.html#bottle.BaseRequest.POST) | no | yes | yes |

文件上传[¶](http://bottle.zzir.cn/index.html#file-uploads)

为了支持文件上传，我们需要小改一下上面 <form> 标签，加上 enctype="multipart/form-data" 属性，告诉浏览器用另一种方式编码表单数据。接下来，我们添加 <input type="file" /> 标签，让用户可以选择需要上传的文件。例子如下。

<**form** action="/upload" method="post" enctype="multipart/form-data">

Category: <**input** type="text" name="category" />

Select a file: <**input** type="file" name="upload" />

<**input** type="submit" value="Start upload" />

</**form**>

Bottle将上传的文件当做一个 [**FileUpload**](http://bottle.zzir.cn/index.html#bottle.FileUpload) 实例存储在 [**BaseRequest.files**](http://bottle.zzir.cn/index.html#bottle.BaseRequest.files) 中，伴随着一些这次上传的元数据。我们假设你仅是想把上传的文件保存到磁盘中。

**@route**('/upload', method='POST')

**def** do\_upload():

category = request.forms.get('category')

upload = request.files.get('upload')

name, ext = os.path.splitext(upload.filename)

**if** ext **not** **in** ('.png','.jpg','.jpeg'):

**return** 'File extension not allowed.'

save\_path = get\_save\_path\_for\_category(category)

upload.save(save\_path) *# appends upload.filename automatically*

**return** 'OK'

[**FileUpload.filename**](http://bottle.zzir.cn/index.html#bottle.FileUpload.filename) 包含客户端传上来的文件的文件名，但为了防止异常字符带来的bug，这里的文件名已经被处理过。如果你需要未经改动的文件名，看看 [**FileUpload.raw\_filename**](http://bottle.zzir.cn/index.html#bottle.FileUpload.raw_filename) 。

如果你想将文件保存到磁盘，强烈建议你使用 [**FileUpload.save**](http://bottle.zzir.cn/index.html#bottle.FileUpload.save) 方法。它避免了一些常见的错误（例如，它不会覆盖已经存在的文件，除非你告诉它可以覆盖），并且更有效地使用内存。你可以通过 [**FileUpload.file**](http://bottle.zzir.cn/index.html#bottle.FileUpload.file) 来直接访问文件对象，但是要谨慎。

JSON内容[¶](http://bottle.zzir.cn/index.html#json-content)

一些JavaScript或支持REST的客户端会发送 application/json 内容给服务器。如果可用（合法的JSON）， [**BaseRequest.json**](http://bottle.zzir.cn/index.html#bottle.BaseRequest.json) 会包含解析后的数据结构。

原始的请求数据[¶](http://bottle.zzir.cn/index.html#the-raw-request-body)

你可以把 [**BaseRequest.body**](http://bottle.zzir.cn/index.html#bottle.BaseRequest.body) 当做一个file-like 对象来访问。根据内容的长度，以及 [**BaseRequest.MEMFILE\_MAX**](http://bottle.zzir.cn/index.html#bottle.BaseRequest.MEMFILE_MAX) 中的设置，它可以是一个 **BytesIO** 缓存或一个磁盘上的临时文件。无论如何，它都是被缓存的。如果你无需缓存，想直接访问文件流，可看看 request['wsgi.input'] 。

WSGI环境[¶](http://bottle.zzir.cn/index.html#wsgi-environment)

每一个 [**BaseRequest**](http://bottle.zzir.cn/index.html#bottle.BaseRequest) 类的实例都包含一个WSGI环境的字典。最初存储在 [**BaseRequest.environ**](http://bottle.zzir.cn/index.html#bottle.BaseRequest.environ) 中，但request对象也表现的像一个字典。大多数有用的数据都通过特定的方法或属性暴露了出来，但如果你想直接访问 [WSGI环境变量](http://bottle.zzir.cn/WSGIspecification) ，可以这样做:

**@route**('/my\_ip')

**def** show\_ip():

ip = request.environ.get('REMOTE\_ADDR')

*# or ip = request.get('REMOTE\_ADDR')*

*# or ip = request['REMOTE\_ADDR']*

**return** template("Your IP is: {{ip}}", ip=ip)

模板[¶](http://bottle.zzir.cn/index.html#templates)

Bottle内置了一个快速的，强大的模板引擎，称为 [SimpleTemplate 模板引擎](http://bottle.zzir.cn/index.html#document-stpl) 。可通过 [**template()**](http://bottle.zzir.cn/index.html#bottle.template) 函数或 [**view()**](http://bottle.zzir.cn/index.html#bottle.view) 修饰器来渲染一个模板。只需提供模板的名字和传递给模板的变量。下面是一个渲染模板的简单例子:

**@route**('/hello')

**@route**('/hello/<name>')

**def** hello(name='World'):

**return** template('hello\_template', name=name)

这会加载 hello\_template.tpl 模板文件，并提供 name 变量。默认情况，Bottle会在 ./views/ 目录查找模板文件(译者注：或当前目录)。可在 bottle.TEMPLATE\_PATH 这个列表中添加更多的模板路径。

[**view()**](http://bottle.zzir.cn/index.html#bottle.view) 修饰器允许你在回调函数中返回一个字典，并将其传递给模板，和 [**template()**](http://bottle.zzir.cn/index.html#bottle.template) 函数做同样的事情。

**@route**('/hello')

**@route**('/hello/<name>')

**@view**('hello\_template')

**def** hello(name='World'):

**return** dict(name=name)

**语法**

模板语法类似于Python的语法。它要确保语句块的正确缩进，所以你在写模板的时候无需担心会出现缩进问题。详细的语法描述可看 [SimpleTemplate 模板引擎](http://bottle.zzir.cn/index.html#document-stpl) 。

简单的模板例子:

%if name == 'World':

<**h1**>Hello {{name}}!</**h1**>

<**p**>This is a test.</**p**>

%else:

<**h1**>Hello {{name.title()}}!</**h1**>

<**p**>How are you?</**p**>

%end

**缓存**

模板在经过编译后被缓存在内存里。你在修改模板文件后，要调用 bottle.TEMPLATES.clear() 函数清除缓存才能看到效果。在debug模式下，缓存被禁用了，无需手动清除缓存。

插件[¶](http://bottle.zzir.cn/index.html#plugins)

*0.9 新版功能.*

Bottle的核心功能覆盖了常见的使用情况，但是作为一个迷你框架，它有它的局限性。所以我们引入了插件机制，插件可以给框架添加其缺少的功能，集成第三方的库，或是自动化一些重复性的工作。

我们有一个不断增长的 [可用插件列表](http://bottle.zzir.cn/index.html#document-plugins/index) 插件列表，大多数插件都被设计为可插拔的。有很大可能，你的问题已经被解决，而且已经有现成的插件可以使用了。如果没有现成的插件， [插件开发指南](http://bottle.zzir.cn/index.html#document-plugindev) 有介绍如何开发一个插件。

插件扮演着各种各样的角色。例如， SQLitePlugin 插件给每个route的回调函数都添加了一个 db 参数，在回调函数被调用的时候，会新建一个数据库连接。这样，使用数据库就非常简单了。

**from** **bottle** **import** route, install, template

**from** **bottle\_sqlite** **import** SQLitePlugin

install(SQLitePlugin(dbfile='/tmp/test.db'))

**@route**('/show/<post\_id:int>')

**def** show(db, post\_id):

c = db.execute('SELECT title, content FROM posts WHERE id = ?', (post\_id,))

row = c.fetchone()

**return** template('show\_post', title=row['title'], text=row['content'])

**@route**('/contact')

**def** contact\_page():

*''' This callback does not need a db connection. Because the 'db'*

*keyword argument is missing, the sqlite plugin ignores this callback*

*completely. '''*

**return** template('contact')

其它插件或许在线程安全的 **local** 对象里面发挥作用，改变 [**request**](http://bottle.zzir.cn/index.html#bottle.request) 对象的细节，过滤回调函数返回的数据或完全绕开回调函数。举个例子，一个用于登录验证的插件会在调用原先的回调函数响应请求之前，验证用户的合法性，如果是非法访问，则返回登录页面而不是调用回调函数。具体的做法要看插件是如何实现的。

整个应用的范围内安装插件[¶](http://bottle.zzir.cn/index.html#application-wide-installation)

可以在整个应用的范围内安装插件，也可以只是安装给某些route。大多数插件都可安全地安装给所有route，也足够智能，可忽略那些并不需要它们的route。

让我们拿 SQLitePlugin 插件举例，它只会影响到那些需要数据库连接的route，其它route都被忽略了。正因为如此，我们可以放心地在整个应用的范围内安装这个插件。

调用 **install()** 函数来安装一个插件:

**from** **bottle\_sqlite** **import** SQLitePlugin

install(SQLitePlugin(dbfile='/tmp/test.db'))

插件没有马上应用到所有route上面，它被延迟执行来确保没有遗漏任何route。你可以先安装插件，再添加route。有时，插件的安装顺序很重要，如果另外一个插件需要连接数据库，那么你就需要先安装操作数据库的插件。

**卸载插件**

调用 **uninstall()** 函数来卸载已经安装的插件

sqlite\_plugin = SQLitePlugin(dbfile='/tmp/test.db')

install(sqlite\_plugin)

uninstall(sqlite\_plugin) *# uninstall a specific plugin*

uninstall(SQLitePlugin) *# uninstall all plugins of that type*

uninstall('sqlite') *# uninstall all plugins with that name*

uninstall(True) *# uninstall all plugins at once*

在任何时候，插件都可以被安装或卸载，即使是在服务器正在运行的时候。一些小技巧应用到了这个特征，例如在需要的时候安装一些供debug和性能测试的插件，但不可滥用这个特性。每一次安装或卸载插件的时候，route缓存都会被刷新，所有插件被重新加载。

注解

模块层面的 **install()** 和 **unistall()** 函数会影响 [默认应用](http://bottle.zzir.cn/index.html#default-app) 。针对应用来管理插件，可使用 [**Bottle**](http://bottle.zzir.cn/index.html#bottle.Bottle) 应用对象的相应方法。

安装给特定的route[¶](http://bottle.zzir.cn/index.html#route-specific-installation)

[**route()**](http://bottle.zzir.cn/index.html#bottle.route) 修饰器的 apply 参数可以给指定的route安装插件

sqlite\_plugin = SQLitePlugin(dbfile='/tmp/test.db')

**@route**('/create', apply=[sqlite\_plugin])

**def** create(db):

db.execute('INSERT INTO ...')

插件黑名单[¶](http://bottle.zzir.cn/index.html#blacklisting-plugins)

如果你想显式地在一些route上面禁用某些插件，可使用 [**route()**](http://bottle.zzir.cn/index.html#bottle.route) 修饰器的 skip 参数:

sqlite\_plugin = SQLitePlugin(dbfile='/tmp/test1.db')

install(sqlite\_plugin)

dbfile1 = '/tmp/test1.db'

dbfile2 = '/tmp/test2.db'

**@route**('/open/<db>', skip=[sqlite\_plugin])

**def** open\_db(db):

*# The 'db' keyword argument is not touched by the plugin this time.*

*# The plugin handle can be used for runtime configuration, too.*

**if** db == 'test1':

sqlite\_plugin.dbfile = dbfile1

**elif** db == 'test2':

sqlite\_plugin.dbfile = dbfile2

**else**:

abort(404, "No such database.")

**return** "Database File switched to: " + sqlite\_plugin.dbfile

skip 参数接受单一的值或是一个list。你可使用插件的名字，类，实例来指定你想要禁用的插件。如果 skip 的值为True，则禁用所有插件。

插件和子应用[¶](http://bottle.zzir.cn/index.html#plugins-and-sub-applications)

大多数插件只会影响到安装了它们的应用。因此，它们不应该影响通过 [**Bottle.mount()**](http://bottle.zzir.cn/index.html#bottle.Bottle.mount) 方法挂载上来的子应用。这里有一个例子。

root = Bottle()

root.mount('/blog', apps.blog)

**@root.route**('/contact', template='contact')

**def** contact():

**return** {'email': 'contact@example.com'}

root.install(plugins.WTForms())

在你挂载一个应用的时候，Bottle在主应用上面创建一个代理route，将所有请求转接给子应用。在代理route上，默认禁用了插件。如上所示，我们的 WTForms 插件影响了 /contact route，但不会影响挂载在root上面的 /blog 。

这个是一个合理的行为，但可被改写。下面的例子，在指定的代理route上面应用了插件。

root.mount('/blog', apps.blog, skip=None)

这里存在一个小难题: 插件会整个子应用当作一个route看待，即是上面提及的代理route。如果想在子应用的每个route上面应用插件，你必须显式地在子应用上面安装插件。

开发[¶](http://bottle.zzir.cn/index.html#development)

到目前为止，你已经学到一些开发的基础，并想写你自己的应用了吧？这里有一些小技巧可提高你的生产力。

默认应用[¶](http://bottle.zzir.cn/index.html#default-application)

Bottle维护一个全局的 [**Bottle**](http://bottle.zzir.cn/index.html#bottle.Bottle) 实例的栈，模块层面的函数和修饰器使用栈顶实例作为默认应用。例如 [**route()**](http://bottle.zzir.cn/index.html#bottle.route) 修饰器，相当于在默认应用上面调用了 [**Bottle.route()**](http://bottle.zzir.cn/index.html#bottle.Bottle.route) 方法。

**@route**('/')

**def** hello():

**return** 'Hello World'

run()

对于小应用来说，这样非常方便，可节约你的工作量。但这同时意味着，在你的模块导入的时候，你定义的 route 就被安装到全局的默认应用中了。为了避免这种模块导入的副作用，Bottle 提供了另外一种方法，显式地管理应用：

app = Bottle()

**@app.route**('/')

**def** hello():

**return** 'Hello World'

app.run()

分离应用对象，大大提高了可重用性。其他开发者可安全地从你的应用中导入 app 对象，然后通过 [**Bottle.mount()**](http://bottle.zzir.cn/index.html#bottle.Bottle.mount) 方法来合并到其它应用中。

*0.13 新版功能.*

从 bottle-0.13 开始，你可以使用 [**Bottle**](http://bottle.zzir.cn/index.html#bottle.Bottle) 实例作为上下文管理器。

app = Bottle()

**with** app:

*# Our application object is now the default*

*# for all shortcut functions and decorators*

**assert** my\_app **is** default\_app()

**@route**('/')

**def** hello():

**return** 'Hello World'

*# Also useful to capture routes defined in other modules*

**import** **some\_package.more\_routes**

调试模式[¶](http://bottle.zzir.cn/index.html#debug-mode)

在开发的早期阶段，调试模式非常有用。

bottle.debug(True)

在调试模式下，当错误发生的时候，Bottle会提供更多的调试信息。同时禁用一些可能妨碍你的优化措施，检查你的错误设置。

下面是调试模式下会发生改变的东西，但这份列表不完整:

* 默认的错误页面会打印出运行栈。
* 模板不会被缓存。
* 插件马上生效。

请确保不要在生产环境中使用调试模式。

自动加载[¶](http://bottle.zzir.cn/index.html#auto-reloading)

在开发的时候，你需要不断地重启服务器来验证你最新的改动。自动加载功能可以替你做这件事情。在你编辑完一个模块文件后，它会自动重启服务器进程，加载最新版本的代码。

**from** **bottle** **import** run

run(reloader=True)

它的工作原理，主进程不会启动服务器，它使用相同的命令行参数，创建一个子进程来启动服务器。请注意，所有模块级别的代码都被执行了至少两次。

子进程中 os.environ['BOOTLE\_CHILD'] 变量的值被设为 True ，它运行一个不会自动加载的服务器。在代码改变后，主进程会终止掉子进程，并创建一个新的子进程。更改模板文件不会触发自动重载，请使用debug模式来禁用模板缓存。

自动加载需要终止子进程。如果你运行在Windows等不支持 signal.SIGINT (会在Python中raise KeyboardInterrupt 异常)的系统上，会使用 signal.SIGTERM 来杀掉子进程。在子进程被 SIGTERM 杀掉的时候，exit handlers和finally等语句不会被执行。

命令行接口[¶](http://bottle.zzir.cn/index.html#command-line-interface)

从0.10版本开始，你可像一个命令行工具那样使用Bottle:

**$** python -m bottle

Usage: bottle.py [options] package.module:app

Options:

-h, --help show this help message and exit

--version show version number.

-b ADDRESS, --bind=ADDRESS

bind socket to ADDRESS.

-s SERVER, --server=SERVER

use SERVER as backend.

-p PLUGIN, --plugin=PLUGIN

install additional plugin/s.

-c FILE, --conf=FILE load config values from FILE.

-C NAME=VALUE, --param=NAME=VALUE

override config values.

--debug start server in debug mode.

--reload auto-reload on file changes.

*ADDRESS* 参数接受一个IP地址或IP:端口，其默认为 localhost:8080 。其它参数都很好地自我解释了。

插件和应用都通过一个导入表达式来指定。包含了导入的路径(例如: package.module )和模块命名空间内的一个表达式，两者用”:”分开。下面是一个简单例子，详见 [**load()**](http://bottle.zzir.cn/index.html#bottle.load) 。

**#** Grab the 'app' object from the 'myapp.controller' module and

**#** start a paste server on port 80 on all interfaces.

python -m bottle -server paste -bind 0.0.0.0:80 myapp.controller:app

**#** Start a self-reloading development server and serve the global

**#** default application. The routes are defined in 'test.py'

python -m bottle --debug --reload test

**#** Install a custom debug plugin with some parameters

python -m bottle --debug --reload --plugin 'utils:DebugPlugin(exc=True)'' test

**#** Serve an application that is created with 'myapp.controller.make\_app()'

**#** on demand.

python -m bottle 'myapp.controller:make\_app()''

部署[¶](http://bottle.zzir.cn/index.html#deployment)

Bottle默认运行在内置的 [wsgiref](http://docs.python.org/library/wsgiref.html#module-wsgiref.simple_server) 服务器上面。这个单线程的HTTP服务器在开发的时候特别有用，但其性能低下，在服务器负载不断增加的时候也许会是性能瓶颈。

最早的解决办法是让Bottle使用 [paste](http://pythonpaste.org/) 或 [cherrypy](http://www.cherrypy.org/) 等多线程的服务器。

bottle.run(server='paste')

在 [部署](http://bottle.zzir.cn/index.html#document-deployment) 章节中，会介绍更多部署的选择。

词汇表[¶](http://bottle.zzir.cn/index.html#glossary)

callback (回调函数)

Programmer code that is to be called when some external action happens. In the context of web frameworks, the mapping between URL paths and application code is often achieved by specifying a callback function for each URL.

decorator (装饰器)

A function returning another function, usually applied as a function transformation using the @decorator syntax. See [python documentation for function definition](http://docs.python.org/reference/compound_stmts.html#function) for more about decorators.

environ (环境)

A structure where information about all documents under the root is saved, and used for cross-referencing. The environment is pickled after the parsing stage, so that successive runs only need to read and parse new and changed documents.

handler function (处理函数)

A function to handle some specific event or situation. In a web framework, the application is developed by attaching a handler function as callback for each specific URL comprising the application.

source directory (源目录)

The directory which, including its subdirectories, contains all source files for one Sphinx project.

配置文件(初稿)[¶](http://bottle.zzir.cn/index.html#configuration-draft)

警告

这是一个新的API， 可以 [告诉我们](mailto:bottlepy%40googlegroups.com) 你的想法

Bottle应用可以在 [**Bottle.config**](http://bottle.zzir.cn/index.html#bottle.Bottle.config) 这个类似于字典的对象中，存储它们的配置。这个对象在很多方面，影响着Bottle框架和相应的插件。你也可以在这个对象中，存储你的自定义配置。

基础知识[¶](http://bottle.zzir.cn/index.html#configuration-basics)

[**Bottle.config**](http://bottle.zzir.cn/index.html#bottle.Bottle.config) 对象的行为很像一个普通的字典。 所有常见的方法预期工作，举个栗子：

**import** **bottle**

app = bottle.default\_app() *# or bottle.Bottle() if you prefer*

app.config['autojson'] = False *# Turns off the "autojson" feature*

app.config['sqlite.db'] = ':memory:' *# Tells the sqlite plugin which db to use*

app.config['myapp.param'] = 'value' *# Example for a custom config value.*

*# Change many values at once*

app.config.update({

'autojson': False,

'sqlite.db': ':memory:',

'myapp.param': 'value'

})

*# Add default values*

app.config.setdefault('myapp.param2', 'some default')

*# Receive values*

param = app.config['myapp.param']

param2 = app.config.get('myapp.param2', 'fallback value')

*# An example route using configuration values*

**@app.route**('/about', view='about.rst')

**def** about():

email = app.config.get('my.email', 'nomail@example.com')

**return** {'email': email}

app对象不一定总是可用的，但只要你在处理一个请求，你可以使用 *request* 对象来获得当前的应用对象和它的配置:

**from** **bottle** **import** request

**def** is\_admin(user):

**return** user == request.app.config['myapp.admin\_user']

命名约定[¶](http://bottle.zzir.cn/index.html#naming-convention)

方便起见，插件和应用应该遵循一些简单的规则，特别是在给配置参数命名的时候:

* 所有的key都应该是小写的字符串，并符合Python的变量命名规则(除了下划线外，没有特殊字符)。
* 命名空间通过点来区分(例如： namespace.field 或 namespace.subnamespacew.field )。
* Bottle框架，使用根命名空间来存储它的配置。插件应该在它们自己的命名空间中存储它们的变量(例如： *sqlite.db* 或 werkzeug.use\_debugger )。
* 你的应用应该使用一个独立的命名空间(例如： myapp.\* )。

从文件中加载配置[¶](http://bottle.zzir.cn/index.html#loading-configuration-from-a-file)

在你不想通过修改代码来修改配置的时候，配置文件是非常有用的。常见的配置文件语法如下:

**[sqlite]**

db = /tmp/test.db

commit = auto

**[myapp]**

admin\_user = defnull

通过 [**ConfigDict.load\_config()**](http://bottle.zzir.cn/index.html#bottle.ConfigDict.load_config) 方法，你可以从一些ini文件中导入配置:

app.config.load\_config('/etc/myapp.conf')

从字典中加载配置[¶](http://bottle.zzir.cn/index.html#loading-configuration-from-a-nested-dict)

另外一个有用的方法，是 [**ConfigDict.load\_dict()**](http://bottle.zzir.cn/index.html#bottle.ConfigDict.load_dict) 。将字典中的配置，放到各自的命名空间下面:

*# Load an entire dict structure*

app.config.load\_dict({

'autojson': False,

'sqlite': { 'db': ':memory:' },

'myapp': {

'param': 'value',

'param2': 'value2'

}

})

**assert** app.config['myapp.param'] == 'value'

*# Load configuration from a json file*

**with** open('/etc/myapp.json') **as** fp:

app.config.load\_dict(json.load(fp))

监听配置的变更[¶](http://bottle.zzir.cn/index.html#listening-to-configuration-changes)

每次 [**Bottle.config**](http://bottle.zzir.cn/index.html#bottle.Bottle.config) 中的值有变更的时候，会触发 config 这个钩子。这个钩子可以用于在运行时，对配置的改动做出响应，例如连接到一个新的数据库，改变后台服务的debug配置，或更改线程池的大小。这个钩子接收两个参数(key, new\_value)，在 [**Bottle.config**](http://bottle.zzir.cn/index.html#bottle.Bottle.config) 中的值被改动之前触发。如果这个钩子抛出了异常，那么 [**Bottle.config**](http://bottle.zzir.cn/index.html#bottle.Bottle.config) 中的值将不会被改动。

**@app.hook**('config')

**def** on\_config\_change(key, value):

**if** key == 'debug':

switch\_own\_debug\_mode\_to(value)

这个钩子不能 *改变* 将要存到 [**Bottle.config**](http://bottle.zzir.cn/index.html#bottle.Bottle.config) 对象中的值。做这件事的是filter(过滤器)。

过滤器和其它元数据[¶](http://bottle.zzir.cn/index.html#filters-and-other-meta-data)

[**ConfigDict**](http://bottle.zzir.cn/index.html#bottle.ConfigDict) 对象允许你给配置中每个key定义元数据。当前定义了help和filter:

help

一个描述字符串。可以被debug或管理工具利用，来帮助网站管理员填写配置。

filter

一个可运行的对象，接受和返回一个值。如果一个key定义了一个filter，任何将要存到这个key中的值，都会先传给filter的相应回调函数。在回调函数中，可做类型转换，有效性检验等工作。

这个功能比较适合被插件使用。它们可以检查它们的配置参数，或触发其它动作，或在 help 字段中，给配置添加说明:

**class** **SomePlugin**(object):

**def** setup(app):

app.config.meta\_set('some.int', 'filter', int)

app.config.meta\_set('some.list', 'filter',

**lambda** val: str(val).split(';'))

app.config.meta\_set('some.list', 'help',

'A semicolon separated list.')

**def** apply(self, callback, route):

...

**import** **bottle**

app = bottle.default\_app()

app.install(SomePlugin())

app.config['some.list'] = 'a;b;c' *# Actually stores ['a', 'b', 'c']*

app.config['some.int'] = 'not an int' *# raises ValueError*

API 文档[¶](http://bottle.zzir.cn/index.html#api-documentation)

*class* **ConfigDict**[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#ConfigDict)[¶](http://bottle.zzir.cn/index.html#bottle.ConfigDict)

A dict-like configuration storage with additional support for namespaces, validators, meta-data, on\_change listeners and more.

**load\_module**(*path*, *squash*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#ConfigDict.load_module)[¶](http://bottle.zzir.cn/index.html#bottle.ConfigDict.load_module)

Load values from a Python module. :param squash: Squash nested dicts into namespaces by using

load\_dict(), otherwise use update()

Example: load\_config(‘my.app.settings’, True) Example: load\_config(‘my.app.settings’, False)

**load\_config**(*filename*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#ConfigDict.load_config)[¶](http://bottle.zzir.cn/index.html#bottle.ConfigDict.load_config)

Load values from an \*.ini style config file.

If the config file contains sections, their names are used as namespaces for the values within. The two special sections DEFAULT and bottle refer to the root namespace (no prefix).

**load\_dict**(*source*, *namespace=''*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#ConfigDict.load_dict)[¶](http://bottle.zzir.cn/index.html#bottle.ConfigDict.load_dict)

Load values from a dictionary structure. Nesting can be used to represent namespaces.

**>>>** c = ConfigDict()

**>>>** c.load\_dict({'some': {'namespace': {'key': 'value'} } })

{'some.namespace.key': 'value'}

**update**(*\*a*, *\*\*ka*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#ConfigDict.update)[¶](http://bottle.zzir.cn/index.html#bottle.ConfigDict.update)

If the first parameter is a string, all keys are prefixed with this namespace. Apart from that it works just as the usual dict.update(). Example: update('some.namespace', key='value')

**meta\_get**(*key*, *metafield*, *default=None*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#ConfigDict.meta_get)[¶](http://bottle.zzir.cn/index.html#bottle.ConfigDict.meta_get)

Return the value of a meta field for a key.

**meta\_set**(*key*, *metafield*, *value*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#ConfigDict.meta_set)[¶](http://bottle.zzir.cn/index.html#bottle.ConfigDict.meta_set)

Set the meta field for a key to a new value.

**meta\_list**(*key*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#ConfigDict.meta_list)[¶](http://bottle.zzir.cn/index.html#bottle.ConfigDict.meta_list)

Return an iterable of meta field names defined for a key.

URL映射[¶](http://bottle.zzir.cn/index.html#request-routing)

Bottle内置一个强大的route引擎，可以给每个浏览器请求找到正确的回调函数。 [tutorial](http://bottle.zzir.cn/index.html#tutorial-routing) 中已经介绍了一些基础知识。接下来是一些进阶知识和route的规则。

route的语法[¶](http://bottle.zzir.cn/index.html#rule-syntax)

**Router** 类中明确区分两种类型的route： **静态route** (例如 /contact )和 **动态route** (例如 /hello/<name> )。包含了 *通配符* 的route即是动态route，除此之外的都是静态的。

*在 0.10 版更改.*

包含通配符，最简单的形式就是将其放到一对<>里面(例如 <name> )。在同一个route里面，这个变量名需要是唯一的。因为稍后会将其当作参数传给回调函数，所以这个变量名的第一个字符应该是字母。

每一个通配符匹配一个或多个字符，直到遇到 / 。类似于 [^/]+ 这样一个正则表达式，确保在route包含多个通配符的时候不出现歧义。

/<action>/<item> 这个规则匹配的情况如下

| **路径** | **结果** |
| --- | --- |
| /save/123 | {'action': 'save', 'item': '123'} |
| /save/123/ | 不匹配 |
| /save/ | 不匹配 |
| //123 | 不匹配 |

你可通过过滤器来改变这一行为，稍后会介绍。

通配符过滤器[¶](http://bottle.zzir.cn/index.html#wildcard-filters)

*0.10 新版功能.*

过滤器被用于定义更特殊的通配符，可在URL中”被匹配到的部分”被传递给回调函数之前，处理其内容。可通过 <name:filter> 或 <name:filer:config> 这样的语句来声明一个过滤器。”config”部分的语法由被使用的过滤器决定。

Bottle中已实现以下过滤器:

* **:int** 匹配一个整形数，并将其转换为int
* **:float** 同上，匹配一个浮点数
* **:path** 匹配所有字符，包括’/’
* **:re[:exp]** 允许在exp中写一个正则表达式

你可在route中添加自己写的过滤器。过滤器是一个有三个返回值的函数：一个正则表达式，一个callable的对象(转换URL片段为Python对象)，另一个callable对象(转换Python对象为URL片段)。过滤器仅接受一个参数，就是设置字符串(译者注：例如re过滤器的exp部分)。

app = Bottle()

**def** list\_filter(config):

*''' Matches a comma separated list of numbers. '''*

delimiter = config **or** ','

regexp = r'\d+(*%s*\d)\*' % re.escape(delimiter)

**def** to\_python(match):

**return** map(int, match.split(delimiter))

**def** to\_url(numbers):

**return** delimiter.join(map(str, numbers))

**return** regexp, to\_python, to\_url

app.router.add\_filter('list', list\_filter)

**@app.route**('/follow/<ids:list>')

**def** follow\_users(ids):

**for** id **in** ids:

...

旧语法[¶](http://bottle.zzir.cn/index.html#legacy-syntax)

*在 0.10 版更改.*

在 **Bottle 0.10** 版本中引入了新的语法，来简单化一些常见用例，但依然兼容旧的语法。新旧语法的区别如下。

| **旧语法** | **新语法** |
| --- | --- |
| :name | <name> |
| :name#regexp# | <name:re:regexp> |
| :#regexp# | <:re:regexp> |
| :## | <:re> |

请尽量在新项目中避免使用旧的语法，虽然它现在还没被废弃，但终究会的。

显式的route配置[¶](http://bottle.zzir.cn/index.html#explicit-routing-configuration)

route修饰器也可以直接当作函数来调用。在复杂的部署中，这种方法或许更灵活，直接由你来控制“何时”及“如何”配置route。

下面是一个简单的例子

**def** setup\_routing():

bottle.route('/', 'GET', index)

bottle.route('/edit', ['GET', 'POST'], edit)

实际上，bottle可以是任何 **Bottle** 类的实例

**def** setup\_routing(app):

app.route('/new', ['GET', 'POST'], form\_new)

app.route('/edit', ['GET', 'POST'], form\_edit)

app = Bottle()

setup\_routing(app)

SimpleTemplate 模板引擎[¶](http://bottle.zzir.cn/index.html#simpletemplate-engine)

Bottle自带了一个快速，强大，易用的模板引擎，名为 *SimpleTemplate* 或简称为 *stpl* 。它是 [**view()**](http://bottle.zzir.cn/index.html#bottle.view) 和 [**template()**](http://bottle.zzir.cn/index.html#bottle.template) 两个函数默认调用的模板引擎。接下来会介绍该引擎的模板语法和一些常见用例。

**基础API :**

[**SimpleTemplate**](http://bottle.zzir.cn/index.html#bottle.SimpleTemplate) 类实现了 [**BaseTemplate**](http://bottle.zzir.cn/index.html#bottle.BaseTemplate) 接口

**>>> from** **bottle** **import** SimpleTemplate

**>>>** tpl = SimpleTemplate('Hello {{name}}!')

**>>>** tpl.render(name='World')

u'Hello World!'

简单起见，我们在例子中使用 [**template()**](http://bottle.zzir.cn/index.html#bottle.template) 函数

**>>> from** **bottle** **import** template

**>>>** template('Hello {{name}}!', name='World')

u'Hello World!'

You can also pass a dictionary into the template using keyword arguments:

**>>> from** **bottle** **import** template

**>>>** my\_dict={'number': '123', 'street': 'Fake St.', 'city': 'Fakeville'}

**>>>** template('I live at {{number}} {{street}}, {{city}}', \*\*my\_dict)

u'I live at 123 Fake St., Fakeville'

注意，编译模板和渲染模板是两件事情，尽管 [**template()**](http://bottle.zzir.cn/index.html#bottle.template) 函数隐藏了这一事实。通常，模板只会被编译一次，然后会被缓存起来，但是会根据不同的参数，被多次渲染。

[**SimpleTemplate**](http://bottle.zzir.cn/index.html#bottle.SimpleTemplate) 的语法[¶](http://bottle.zzir.cn/index.html#simpletemplate-syntax)

虽然Python是一门强大的语言，但它对空白敏感的语法令其很难作为一个模板语言。SimpleTemplate移除了一些限制，允许你写出干净的，有可读性的，可维护的模板，且保留了Python的强大功能。

警告

[**SimpleTemplate**](http://bottle.zzir.cn/index.html#bottle.SimpleTemplate) 模板会被编译为 Python 字节码，且在每次在 [**SimpleTemplate.render()**](http://bottle.zzir.cn/index.html#bottle.SimpleTemplate.render) 渲染的时候执行。 禁止渲染不可信的模板! 它们可能包含恶意代码。

内嵌表达式[¶](http://bottle.zzir.cn/index.html#inline-expressions)

你已经在上面 "Hello World" 的例子中学到了 {{...}}语句的用法 。只要在 {{...}} 中的 Python 语句返回一个字符串或有一个字符串的表达形式，它就是一个有效的语句。

**>>>** template('Hello {{name}}!', name='World')

u'Hello World!'

**>>>** template('Hello {{name.title() if name else "stranger"}}!', name=None)

u'Hello stranger!'

**>>>** template('Hello {{name.title() if name else "stranger"}}!', name='mArC')

u'Hello Marc!'

{{}}中的Python语句会在渲染的时候被执行，可访问传递给 [**SimpleTemplate.render()**](http://bottle.zzir.cn/index.html#bottle.SimpleTemplate.render) 方法的所有参数。默认情况下，自动转义HTML标签以防止 [XSS](http://en.wikipedia.org/wiki/Cross-Site_Scripting) 攻击。可在语句前加上”!”来关闭自动转义。

**>>>** template('Hello {{name}}!', name='<b>World</b>')

u'Hello &lt;b&gt;World&lt;/b&gt;!'

**>>>** template('Hello {{!name}}!', name='<b>World</b>')

u'Hello <b>World</b>!'

嵌入Pyhton代码[¶](http://bottle.zzir.cn/index.html#embedded-python-code)

模板引擎允许你在模板中嵌入代码行和代码块。 代码行以 % 开头，代码块包含在 <% 和 %> 之间：

% name = "Bob" # a line of python code

<p>Some plain text in between</p>

<%

# A block of python code

name = name.title().strip()

%>

<p>More plain text</p>

嵌入的代码遵循 Python 语法，但有两个额外的语法规则：

* **忽略行首缩进。** 你可以尽可能多的把空白符写在语句前面。 为提高可读性，可将你的代码与周围的标记对齐。
* 代码块嵌入通常以 end 关键字显式结束。

<**ul**>

% for item in basket:

<**li**>{{item}}</**li**>

% end

</**ul**>

无论 % 还是 <% 标记只识别每行第一个非空白字符。 不要在文字中间漏掉出现的模板标记，只有在模板标记每行开始才能忽略反斜杠。少数情况下反斜杠 + 模板标记出现在一行的开始， 用以获得内联表达式的字符串值：

This line contains % and <% but no python code.

\% This text-line starts with the '%' token.

\<% Another line that starts with a token but is rendered as text.

{{'\\%'}} this line starts with an escaped token.

如果发现漏了一些, 可以考略使用 自定义标记。

空白符控制[¶](http://bottle.zzir.cn/index.html#whitespace-control)

代码块和代码行占用整行。 代码段前后的空白符将被删除。由于是代码嵌入你不会在模板中看到空行和结尾有空白字符出现：

<**div**>

% if True:

<**span**>content</**span**>

% end

</**div**>

处理后的HTML代码片段：

<**div**>

<**span**>content</**span**>

</**div**>

嵌入的代码会从新行开始，如果你不想模板以这种方式呈现。要忽略换行符，文本行双反斜杠结尾即可：

<**div**>\\

%if True:

<**span**>content</**span**>\\

%end

</**div**>

这次模板呈现方式如下：

<**div**><**span**>content</**span**></**div**>

仅适用在代码段之前。在其它地方你可以自由的控制空白符而不需要特殊语法。

模板函数[¶](http://bottle.zzir.cn/index.html#template-functions)

每一个模板都会通过一系列函数预先加载，这样有助于在大多数场景下的使用。这些函数始终有效，你不用导入它们，一切不在这里的还有非常好用的 Python 库。你可以在模板中 import它们。毕竟它们都是 Python。

*在 0.12 版更改:* 之前的版本， [**include()**](http://bottle.zzir.cn/index.html#stpl.include) 和 [**rebase()**](http://bottle.zzir.cn/index.html#stpl.rebase) 只是关键字还不是函数。

**include**(*sub\_template*, *\*\*variables*)[¶](http://bottle.zzir.cn/index.html#stpl.include)

渲染一个子模版并指定变量生成的内容到当前模板。该函数返回一个字典，该字典包含了子模板中定义的局部变量：

% include('header.tpl', title='Page Title')

Page Content

% include('footer.tpl')

**rebase**(*name*, *\*\*variables*)[¶](http://bottle.zzir.cn/index.html#stpl.rebase)

把当前模板纳入到其它模板。在当前模板渲染之后，其生成的内容存储在变量 base 中并传递给基模板，然后再呈现。这可以用来 *wrap* 一个模板与周围的内容，或从其它模板引擎中继承：

% rebase('base.tpl', title='Page Title')

<**p**>Page Content ...</**p**>

如 base.tpl:

<html>

<head>

<title>{{title or 'No title'}}</title>

</head>

<body>

{{!base}}

</body>

</html>

在模板中访问未定义的变量会引起 **NameError** 并停止渲染。这是标准的 Python 行为而且屡见不鲜，但普通的 Python 缺少一个简单的方法来检查变量的可用性。如果你想支持灵活的输入或使用相同的模板在不同的情况。这些功能可以帮助：

**defined**(*name*)[¶](http://bottle.zzir.cn/index.html#stpl.defined)

如果变量已定义则返回True，反之返回False。

**get**(*name*, *default=None*)[¶](http://bottle.zzir.cn/index.html#stpl.get)

返回该变量，或一个默认值

**setdefault**(*name*, *default*)[¶](http://bottle.zzir.cn/index.html#stpl.setdefault)

如果该变量未定义，则定义它，赋一个默认值，返回该变量

下面是使用了这三个函数的例子，实现了模板中的可选参数。

% setdefault('text', 'No Text')

<**h1**>{{get('title', 'No Title')}}</**h1**>

<**p**> {{ text }} </**p**>

% if defined('author'):

<**p**>By {{ author }}</**p**>

% end

[**SimpleTemplate**](http://bottle.zzir.cn/index.html#bottle.SimpleTemplate) API[¶](http://bottle.zzir.cn/index.html#simpletemplate-api)

*class* **SimpleTemplate**(*source=None*, *name=None*, *lookup=None*, *encoding='utf8'*, *\*\*settings*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#SimpleTemplate)[¶](http://bottle.zzir.cn/index.html#bottle.SimpleTemplate)

**render**(*\*args*, *\*\*kwargs*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#SimpleTemplate.render)[¶](http://bottle.zzir.cn/index.html#bottle.SimpleTemplate.render)

渲染模板使用关键字参数作为局部变量。

部署[¶](http://bottle.zzir.cn/index.html#deployment)

不添加任何参数，直接运行Bottle的 **run()** 函数，会启动一个本地的开发服务器，监听8080端口。你可在同一部主机上访问http://localhost:8080/来测试你的应用。

可更改服务器监听的IP地址(例如： run(host='192.168.0.1') )，来让应用对其可访问，或者让服务器接受所有地址的请求(例如： run(host='0.0.0.0') )。可通过类似的方法改变服务器监听的端口，但如果你选择了一个小于1024的端口，则需要root权限。HTTP服务器的标准端口是80端口:

run(host='0.0.0.0', port=80) # Listen to HTTP requests on all interfaces

可选服务器[¶](http://bottle.zzir.cn/index.html#server-options)

内置的服务器基于 [wsgiref WSGIServer](http://docs.python.org/library/wsgiref.html#module-wsgiref.simple_server) 。这个单线程的HTTP服务器很适合用于开发，但当服务器的负载上升的时候，会成为一个性能瓶颈。有三种方法可消除这一瓶颈：

* 使用多线程或异步的服务器
* 运行多个服务器，使用负载均衡
* 同时使用上面两种方法

**多线程 (Multi-thread)** 服务是 ‘classic’ 的方法，他们非常稳健，能实现合理的速度并且易于管理。但有一个缺点，它们在同一时间只能处理有限数量的连接和只使用一个CPU核心，这是因为 Python 在运行时的 “Global Interpreter Lock” (全局解释器锁 GIL)。这并不损害大多数应用程序，不管怎么说大多数时间它们在等待网络IO, 但可能会减慢 CPU 密集型任务 （例如图像处理）。

**异步 (Asynchronous)** 模式非常快，可以处理几乎无限数量的并发连接，并且易于管理。为了充分利用他们的性能，您需要相应地设计您的应用程序和理解特定服务器的概念。

**多进程** (Forking) 服务器就没有受到GIL的限制，能利用多个CPU核心，但服务器实例之间的交流代价比较高昂。你需要一个数据库或消息队列来在进程之间共享状态，或将你的应用设计成根本不需要共享状态。多进程服务器的安装也比较负责，但已经有很多好的教程了。

更改服务器后端[¶](http://bottle.zzir.cn/index.html#switching-the-server-backend)

提高性能的最简单方法是安装一个多线程服务器库像 [paste](http://pythonpaste.org/) 或 [cherrypy](http://www.cherrypy.org/) 并告诉 Bottle 使用，而不是默认的单线程服务器：

bottle.run(server='paste')

Bottle为很多常见的WSGI服务器都编写了适配器，能自动化安装过程。下面是一个不完整的清单:

| **名称** | **主页** | **描述** |
| --- | --- | --- |
| cgi |  | Run as CGI script |
| flup | [flup](http://trac.saddi.com/flup) | Run as FastCGI process |
| gae | [gae](http://code.google.com/appengine/docs/python/overview.html) | 用于Google App Engine |
| wsgiref | [wsgiref](http://docs.python.org/library/wsgiref.html) | 默认的单线程服务器 |
| cherrypy | [cherrypy](http://www.cherrypy.org/) | 多线程，稳定 |
| paste | [paste](http://pythonpaste.org/) | 多线程，稳定，久经考验，充分测试 |
| rocket | [rocket](http://pypi.python.org/pypi/rocket) | 多线程 |
| waitress | [waitress](http://readthedocs.org/docs/waitress/en/latest/) | 多线程，源于Pyramid |
| gunicorn | [gunicorn](http://pypi.python.org/pypi/gunicorn) | Pre-forked, partly written in C |
| eventlet | [eventlet](http://eventlet.net/) | 支持WSGI的异步框架 |
| gevent | [gevent](http://www.gevent.org/) | 异步 (greenlets) |
| diesel | [diesel](http://dieselweb.org/) | 异步 (greenlets) |
| fapws3 | [fapws3](http://www.fapws.org/) | 异步 (network side only), written in C |
| tornado | [tornado](http://www.tornadoweb.org/) | 异步，支撑Facebook的部分应用 |
| twisted | [twisted](http://twistedmatrix.com/) | 异步, well tested but... twisted |
| meinheld | [meinheld](http://pypi.python.org/pypi/meinheld) | 异步，部分用C语言编写 |
| bjoern | [bjoern](http://pypi.python.org/pypi/bjoern) | 异步，用C语言编写，非常快 |
| auto |  | 自动选择一个可用的服务器 |

完整的列表在 **server\_names** 。

如果没有适合你的服务器的适配器，或者你需要更多地控制服务器的安装，你也许需要手动启动服务器。可参考你的服务器的文档，看看是如何运行一个WSGI应用。下面是一个使用 [paste](http://pythonpaste.org/) 的例子。

application = bottle.default\_app()

from paste import httpserver

httpserver.serve(application, host='0.0.0.0', port=80)

Apache mod\_wsgi[¶](http://bottle.zzir.cn/index.html#apache-mod-wsgi)

除了直接在Bottle里面运行HTTP服务器，你也可以将你的应用部署到一个Apache服务器上，使用 [mod\_wsgi](http://code.google.com/p/modwsgi/) 来运行。

你需要的只是提供一个 application 对象的 app.wsgi 文件。mod\_wsgi会使用这个对象来启动你的应用，这个对象必须是兼容WSGI的 callable对象。

File /var/www/yourapp/app.wsgi:

import os

# Change working directory so relative paths (and template lookup) work again

os.chdir(os.path.dirname(\_\_file\_\_))

import bottle

# ... build or import your bottle application here ...

# Do NOT use bottle.run() with mod\_wsgi

application = bottle.default\_app()

Apache的配置

<VirtualHost \*>

ServerName example.com

WSGIDaemonProcess yourapp user=www-data group=www-data processes=1 threads=5

WSGIScriptAlias / /var/www/yourapp/app.wsgi

<Directory /var/www/yourapp>

WSGIProcessGroup yourapp

WSGIApplicationGroup %{GLOBAL}

Order deny,allow

Allow from all

</Directory>

</VirtualHost>

With newer versions of Apache (2.4) use a configuration similar to this:

<VirtualHost \*>

ServerName example.com

WSGIDaemonProcess yourapp user=www-data group=www-data processes=1 threads=5

WSGIScriptAlias / /var/www/yourapp/app.wsgi

<Directory /var/www/yourapp>

WSGIProcessGroup yourapp

WSGIApplicationGroup %{GLOBAL}

Require all granted

</Directory>

</VirtualHost>

Google AppEngine[¶](http://bottle.zzir.cn/index.html#google-appengine)

*0.9 新版功能.*

New App Engine applications using the Python 2.7 runtime environment support any WSGI application and should be configured to use the Bottle application object directly. For example suppose your application’s main module is myapp.py:

import bottle

@bottle.route('/')

def home():

return '<**html**><**head**></**head**><**body**>Hello world!</**body**></**html**>'

app = bottle.default\_app()

Then you can configure App Engine’s app.yaml to use the app object like so:

application: myapp

version: 1

runtime: python27

api\_version: 1

handlers:

- url: /.\*

script: myapp.app

Bottle also provides a gae server adapter for legacy App Engine applications using the Python 2.5 runtime environment. It works similar to the cgi adapter in that it does not start a new HTTP server, but prepares and optimizes your application for Google App Engine and makes sure it conforms to their API:

bottle.run(server='gae') # No need for a host or port setting.

It is always a good idea to let GAE serve static files directly. Here is example for a working app.yaml (using the legacy Python 2.5 runtime environment):

application: myapp

version: 1

runtime: python

api\_version: 1

handlers:

- url: /static

static\_dir: static

- url: /.\*

script: myapp.py

负载均衡 (手动安装)[¶](http://bottle.zzir.cn/index.html#load-balancer-manual-setup)

单一的Python进程一次只能使用一个CPU内核，即使CPU是多核的。我们的方法就是在多核CPU的机器上，使用多线程来实现负载均衡。

不只是启动一个应用服务器，你需要同时启动多个应用服务器，监听不同的端口(localhost:8080, 8081, 8082, ...)。你可选择任何服务器，甚至那些异步服务器。然后一个高性能的负载均衡器，像一个反向代理那样工作，将新的请求发送到一个随机端口，在多个服务器之间分散压力。这样你就可以利用所有的CPU核心，甚至在多个机器上实现负载均衡。

负载平衡器最快的是 [Pound](http://www.apsis.ch/pound) ，但常见的web服务器代理模块也可以很好的工作。

Pound的例子:

ListenHTTP

Address 0.0.0.0

Port 80

Service

BackEnd

Address 127.0.0.1

Port 8080

End

BackEnd

Address 127.0.0.1

Port 8081

End

End

End

Apache的例子:

<Proxy balancer://mycluster>

BalancerMember http://127.0.0.1:8080

BalancerMember http://127.0.0.1:8081

</Proxy>

ProxyPass / balancer://mycluster

Lighttpd的例子:

server.modules += ( "mod\_proxy" )

proxy.server = (

"" => (

"wsgi1" => ( "host" => "127.0.0.1", "port" => 8080 ),

"wsgi2" => ( "host" => "127.0.0.1", "port" => 8081 )

)

)

CGI这个老好人[¶](http://bottle.zzir.cn/index.html#good-old-cgi)

CGI服务器会为每个请求启动一个进程。虽然这样代价高昂，但有时这是唯一的选择。 *cgi* 这个适配器实际上并没有启动一个CGI服务器，只是将你的Bottle应用转换成了一个有效的CGI应用。

bottle.run(server='cgi')

API参考[¶](http://bottle.zzir.cn/index.html#module-bottle)

这份文档几乎是全自动生成的。如果你刚接触bottle，也许 [教程](http://bottle.zzir.cn/index.html#document-tutorial) 会更有帮助。

模块内容[¶](http://bottle.zzir.cn/index.html#module-contents)

模块定义了几个函数、常量和异常。

**debug**(*mode=True*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#debug)[¶](http://bottle.zzir.cn/index.html#bottle.debug)

Change the debug level. There is only one debug level supported at the moment.

**run**(*app=None*, *server='wsgiref'*, *host='127.0.0.1'*, *port=8080*, *interval=1*, *reloader=False*, *quiet=False*, *plugins=None*, *debug=None*, *config=None*, *\*\*kargs*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#run)[¶](http://bottle.zzir.cn/index.html#bottle.run)

Start a server instance. This method blocks until the server terminates.

|  |  |
| --- | --- |
| **参数:** | * **app** – WSGI application or target string supported by [**load\_app()**](http://bottle.zzir.cn/index.html#bottle.load_app). (default: [**default\_app()**](http://bottle.zzir.cn/index.html#bottle.default_app)) * **server** – Server adapter to use. See **server\_names** keys for valid names or pass a **ServerAdapter** subclass. (default: *wsgiref*) * **host** – Server address to bind to. Pass 0.0.0.0 to listens on all interfaces including the external one. (default: 127.0.0.1) * **port** – Server port to bind to. Values below 1024 require root privileges. (default: 8080) * **reloader** – Start auto-reloading server? (default: False) * **interval** – Auto-reloader interval in seconds (default: 1) * **quiet** – Suppress output to stdout and stderr? (default: False) * **options** – Options passed to the server adapter. |

**load**(*target*, *\*\*namespace*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#load)[¶](http://bottle.zzir.cn/index.html#bottle.load)

Import a module or fetch an object from a module.

* package.module returns *module* as a module object.
* pack.mod:name returns the module variable *name* from *pack.mod*.
* pack.mod:func() calls *pack.mod.func()* and returns the result.

The last form accepts not only function calls, but any type of expression. Keyword arguments passed to this function are available as local variables. Example: import\_string('re:compile(x)', x='[a-z]')

**load\_app**(*target*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#load_app)[¶](http://bottle.zzir.cn/index.html#bottle.load_app)

Load a bottle application from a module and make sure that the import does not affect the current default application, but returns a separate application object. See [**load()**](http://bottle.zzir.cn/index.html#bottle.load) for the target parameter.

**request** *= <LocalRequest: GET http://127.0.0.1/>*[¶](http://bottle.zzir.cn/index.html#bottle.request)

A thread-safe instance of [**LocalRequest**](http://bottle.zzir.cn/index.html#bottle.LocalRequest). If accessed from within a request callback, this instance always refers to the *current* request (even on a multi-threaded server).

**response** *= Content-Type: text/html; charset=UTF-8* [¶](http://bottle.zzir.cn/index.html#bottle.response)

A thread-safe instance of [**LocalResponse**](http://bottle.zzir.cn/index.html#bottle.LocalResponse). It is used to change the HTTP response for the *current* request.

**HTTP\_CODES** *= {300: 'Multiple Choices', 429: 'Too Many Requests', 400: 'Bad Request', 401: 'Unauthorized', 402: 'Payment Required', 403: 'Forbidden', 404: 'Not Found', 405: 'Method Not Allowed', 406: 'Not Acceptable', 407: 'Proxy Authentication Required', 408: 'Request Timeout', 409: 'Conflict', 410: 'Gone', 411: 'Length Required', 412: 'Precondition Failed', 413: 'Request Entity Too Large', 414: 'Request-URI Too Long', 415: 'Unsupported Media Type', 416: 'Requested Range Not Satisfiable', 417: 'Expectation Failed', 418: "I'm a teapot", 428: 'Precondition Required', 301: 'Moved Permanently', 302: 'Found', 431: 'Request Header Fields Too Large', 304: 'Not Modified', 305: 'Use Proxy', 306: '(Unused)', 307: 'Temporary Redirect', 200: 'OK', 201: 'Created', 202: 'Accepted', 203: 'Non-Authoritative Information', 204: 'No Content', 205: 'Reset Content', 206: 'Partial Content', 303: 'See Other', 100: 'Continue', 101: 'Switching Protocols', 500: 'Internal Server Error', 501: 'Not Implemented', 502: 'Bad Gateway', 503: 'Service Unavailable', 504: 'Gateway Timeout', 505: 'HTTP Version Not Supported', 511: 'Network Authentication Required'}*[¶](http://bottle.zzir.cn/index.html#bottle.HTTP_CODES)

A dict to map HTTP status codes (e.g. 404) to phrases (e.g. ‘Not Found’)

**app**()[¶](http://bottle.zzir.cn/index.html#bottle.app)

**default\_app**()[¶](http://bottle.zzir.cn/index.html#bottle.default_app)

Return the current [默认应用](http://bottle.zzir.cn/index.html#default-app). Actually, these are callable instances of [**AppStack**](http://bottle.zzir.cn/index.html#bottle.AppStack) and implement a stack-like API.

Routing[¶](http://bottle.zzir.cn/index.html#routing)

Bottle maintains a stack of [**Bottle**](http://bottle.zzir.cn/index.html#bottle.Bottle) instances (see [**app()**](http://bottle.zzir.cn/index.html#bottle.app) and [**AppStack**](http://bottle.zzir.cn/index.html#bottle.AppStack)) and uses the top of the stack as a *default application* for some of the module-level functions and decorators.

**route**(*path*, *method='GET'*, *callback=None*, *\*\*options*)[¶](http://bottle.zzir.cn/index.html#bottle.route)

**get**(*...*)[¶](http://bottle.zzir.cn/index.html#bottle.get)

**post**(*...*)[¶](http://bottle.zzir.cn/index.html#bottle.post)

**put**(*...*)[¶](http://bottle.zzir.cn/index.html#bottle.put)

**delete**(*...*)[¶](http://bottle.zzir.cn/index.html#bottle.delete)

**patch**(*...*)[¶](http://bottle.zzir.cn/index.html#bottle.patch)

Decorator to install a route to the current default application. See [**Bottle.route()**](http://bottle.zzir.cn/index.html#bottle.Bottle.route) for details.

**error**(*...*)[¶](http://bottle.zzir.cn/index.html#bottle.error)

Decorator to install an error handler to the current default application. See [**Bottle.error()**](http://bottle.zzir.cn/index.html#bottle.Bottle.error) for details.

WSGI and HTTP Utilities[¶](http://bottle.zzir.cn/index.html#wsgi-and-http-utilities)

**parse\_date**(*ims*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#parse_date)[¶](http://bottle.zzir.cn/index.html#bottle.parse_date)

Parse rfc1123, rfc850 and asctime timestamps and return UTC epoch.

**parse\_auth**(*header*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#parse_auth)[¶](http://bottle.zzir.cn/index.html#bottle.parse_auth)

Parse rfc2617 HTTP authentication header string (basic) and return (user,pass) tuple or None

**cookie\_encode**(*data*, *key*, *digestmod=None*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#cookie_encode)[¶](http://bottle.zzir.cn/index.html#bottle.cookie_encode)

Encode and sign a pickle-able object. Return a (byte) string

**cookie\_decode**(*data*, *key*, *digestmod=None*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#cookie_decode)[¶](http://bottle.zzir.cn/index.html#bottle.cookie_decode)

Verify and decode an encoded string. Return an object or None.

**cookie\_is\_encoded**(*data*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#cookie_is_encoded)[¶](http://bottle.zzir.cn/index.html#bottle.cookie_is_encoded)

Return True if the argument looks like a encoded cookie.

**yieldroutes**(*func*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#yieldroutes)[¶](http://bottle.zzir.cn/index.html#bottle.yieldroutes)

Return a generator for routes that match the signature (name, args) of the func parameter. This may yield more than one route if the function takes optional keyword arguments. The output is best described by example:

a() -> '/a'

b(x, y) -> '/b/<**x**>/<**y**>'

c(x, y=5) -> '/c/<**x**>' and '/c/<**x**>/<**y**>'

d(x=5, y=6) -> '/d' and '/d/<**x**>' and '/d/<**x**>/<**y**>'

**path\_shift**(*script\_name*, *path\_info*, *shift=1*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#path_shift)[¶](http://bottle.zzir.cn/index.html#bottle.path_shift)

Shift path fragments from PATH\_INFO to SCRIPT\_NAME and vice versa.

|  |  |
| --- | --- |
| **返回:** | The modified paths. |
| **参数:** | * **script\_name** – The SCRIPT\_NAME path. * **script\_name** – The PATH\_INFO path. * **shift** – The number of path fragments to shift. May be negative to change the shift direction. (default: 1) |

Data Structures[¶](http://bottle.zzir.cn/index.html#data-structures)

*class* **MultiDict**(*\*a*, *\*\*k*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#MultiDict)[¶](http://bottle.zzir.cn/index.html#bottle.MultiDict)

This dict stores multiple values per key, but behaves exactly like a normal dict in that it returns only the newest value for any given key. There are special methods available to access the full list of values.

**get**(*key*, *default=None*, *index=-1*, *type=None*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#MultiDict.get)[¶](http://bottle.zzir.cn/index.html#bottle.MultiDict.get)

Return the most recent value for a key.

|  |  |
| --- | --- |
| **参数:** | * **default** – The default value to be returned if the key is not present or the type conversion fails. * **index** – An index for the list of available values. * **type** – If defined, this callable is used to cast the value into a specific type. Exception are suppressed and result in the default value to be returned. |

**append**(*key*, *value*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#MultiDict.append)[¶](http://bottle.zzir.cn/index.html#bottle.MultiDict.append)

Add a new value to the list of values for this key.

**replace**(*key*, *value*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#MultiDict.replace)[¶](http://bottle.zzir.cn/index.html#bottle.MultiDict.replace)

Replace the list of values with a single value.

**getall**(*key*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#MultiDict.getall)[¶](http://bottle.zzir.cn/index.html#bottle.MultiDict.getall)

Return a (possibly empty) list of values for a key.

**getone**(*key*, *default=None*, *index=-1*, *type=None*)[¶](http://bottle.zzir.cn/index.html#bottle.MultiDict.getone)

Aliases for WTForms to mimic other multi-dict APIs (Django)

**getlist**(*key*)[¶](http://bottle.zzir.cn/index.html#bottle.MultiDict.getlist)

Return a (possibly empty) list of values for a key.

*class* **HeaderDict**(*\*a*, *\*\*ka*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#HeaderDict)[¶](http://bottle.zzir.cn/index.html#bottle.HeaderDict)

A case-insensitive version of [**MultiDict**](http://bottle.zzir.cn/index.html#bottle.MultiDict) that defaults to replace the old value instead of appending it.

*class* **FormsDict**(*\*a*, *\*\*k*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#FormsDict)[¶](http://bottle.zzir.cn/index.html#bottle.FormsDict)

This [**MultiDict**](http://bottle.zzir.cn/index.html#bottle.MultiDict) subclass is used to store request form data. Additionally to the normal dict-like item access methods (which return unmodified data as native strings), this container also supports attribute-like access to its values. Attributes are automatically de- or recoded to match [**input\_encoding**](http://bottle.zzir.cn/index.html#bottle.FormsDict.input_encoding) (default: ‘utf8’). Missing attributes default to an empty string.

**input\_encoding** *= 'utf8'*[¶](http://bottle.zzir.cn/index.html#bottle.FormsDict.input_encoding)

Encoding used for attribute values.

**recode\_unicode** *= True*[¶](http://bottle.zzir.cn/index.html#bottle.FormsDict.recode_unicode)

If true (default), unicode strings are first encoded with *latin1* and then decoded to match [**input\_encoding**](http://bottle.zzir.cn/index.html#bottle.FormsDict.input_encoding).

**decode**(*encoding=None*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#FormsDict.decode)[¶](http://bottle.zzir.cn/index.html#bottle.FormsDict.decode)

Returns a copy with all keys and values de- or recoded to match [**input\_encoding**](http://bottle.zzir.cn/index.html#bottle.FormsDict.input_encoding). Some libraries (e.g. WTForms) want a unicode dictionary.

**getunicode**(*name*, *default=None*, *encoding=None*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#FormsDict.getunicode)[¶](http://bottle.zzir.cn/index.html#bottle.FormsDict.getunicode)

Return the value as a unicode string, or the default.

*class* **WSGIHeaderDict**(*environ*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#WSGIHeaderDict)[¶](http://bottle.zzir.cn/index.html#bottle.WSGIHeaderDict)

This dict-like class wraps a WSGI environ dict and provides convenient access to HTTP\_\* fields. Keys and values are native strings (2.x bytes or 3.x unicode) and keys are case-insensitive. If the WSGI environment contains non-native string values, these are de- or encoded using a lossless ‘latin1’ character set.

The API will remain stable even on changes to the relevant PEPs. Currently PEP 333, 444 and 3333 are supported. (PEP 444 is the only one that uses non-native strings.)

**cgikeys** *= ('CONTENT\_TYPE', 'CONTENT\_LENGTH')*[¶](http://bottle.zzir.cn/index.html#bottle.WSGIHeaderDict.cgikeys)

List of keys that do not have a HTTP\_ prefix.

**raw**(*key*, *default=None*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#WSGIHeaderDict.raw)[¶](http://bottle.zzir.cn/index.html#bottle.WSGIHeaderDict.raw)

Return the header value as is (may be bytes or unicode).

*class* **AppStack**[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#AppStack)[¶](http://bottle.zzir.cn/index.html#bottle.AppStack)

A stack-like list. Calling it returns the head of the stack.

**pop**()[¶](http://bottle.zzir.cn/index.html#bottle.AppStack.pop)

Return the current default application and remove it from the stack.

**push**(*value=None*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#AppStack.push)[¶](http://bottle.zzir.cn/index.html#bottle.AppStack.push)

Add a new [**Bottle**](http://bottle.zzir.cn/index.html#bottle.Bottle) instance to the stack

**new\_app**(*value=None*)[¶](http://bottle.zzir.cn/index.html#bottle.AppStack.new_app)

Add a new [**Bottle**](http://bottle.zzir.cn/index.html#bottle.Bottle) instance to the stack

*class* **ResourceManager**(*base='./'*, *opener=<built-in function open>*, *cachemode='all'*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#ResourceManager)[¶](http://bottle.zzir.cn/index.html#bottle.ResourceManager)

This class manages a list of search paths and helps to find and open application-bound resources (files).

|  |  |
| --- | --- |
| **参数:** | * **base** – default value for [**add\_path()**](http://bottle.zzir.cn/index.html#bottle.ResourceManager.add_path) calls. * **opener** – callable used to open resources. * **cachemode** – controls which lookups are cached. One of ‘all’, ‘found’ or ‘none’. |

**path** *= None*[¶](http://bottle.zzir.cn/index.html#bottle.ResourceManager.path)

A list of search paths. See [**add\_path()**](http://bottle.zzir.cn/index.html#bottle.ResourceManager.add_path) for details.

**cache** *= None*[¶](http://bottle.zzir.cn/index.html#bottle.ResourceManager.cache)

A cache for resolved paths. res.cache.clear() clears the cache.

**add\_path**(*path*, *base=None*, *index=None*, *create=False*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#ResourceManager.add_path)[¶](http://bottle.zzir.cn/index.html#bottle.ResourceManager.add_path)

Add a new path to the list of search paths. Return False if the path does not exist.

|  |  |
| --- | --- |
| **参数:** | * **path** – The new search path. Relative paths are turned into an absolute and normalized form. If the path looks like a file (not ending in */*), the filename is stripped off. * **base** – Path used to absolutize relative search paths. Defaults to **base** which defaults to os.getcwd(). * **index** – Position within the list of search paths. Defaults to last index (appends to the list). |

The *base* parameter makes it easy to reference files installed along with a python module or package:

res.add\_path('./resources/', \_\_file\_\_)

**lookup**(*name*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#ResourceManager.lookup)[¶](http://bottle.zzir.cn/index.html#bottle.ResourceManager.lookup)

Search for a resource and return an absolute file path, or *None*.

The [**path**](http://bottle.zzir.cn/index.html#bottle.ResourceManager.path) list is searched in order. The first match is returend. Symlinks are followed. The result is cached to speed up future lookups.

**open**(*name*, *mode='r'*, *\*args*, *\*\*kwargs*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#ResourceManager.open)[¶](http://bottle.zzir.cn/index.html#bottle.ResourceManager.open)

Find a resource and return a file object, or raise IOError.

*class* **FileUpload**(*fileobj*, *name*, *filename*, *headers=None*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#FileUpload)[¶](http://bottle.zzir.cn/index.html#bottle.FileUpload)

**file** *= None*[¶](http://bottle.zzir.cn/index.html#bottle.FileUpload.file)

Open file(-like) object (BytesIO buffer or temporary file)

**name** *= None*[¶](http://bottle.zzir.cn/index.html#bottle.FileUpload.name)

Name of the upload form field

**raw\_filename** *= None*[¶](http://bottle.zzir.cn/index.html#bottle.FileUpload.raw_filename)

Raw filename as sent by the client (may contain unsafe characters)

**headers** *= None*[¶](http://bottle.zzir.cn/index.html#bottle.FileUpload.headers)

A [**HeaderDict**](http://bottle.zzir.cn/index.html#bottle.HeaderDict) with additional headers (e.g. content-type)

**content\_type**[¶](http://bottle.zzir.cn/index.html#bottle.FileUpload.content_type)

Current value of the ‘Content-Type’ header.

**content\_length**[¶](http://bottle.zzir.cn/index.html#bottle.FileUpload.content_length)

Current value of the ‘Content-Length’ header.

**filename**[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#FileUpload.filename)[¶](http://bottle.zzir.cn/index.html#bottle.FileUpload.filename)

Name of the file on the client file system, but normalized to ensure file system compatibility. An empty filename is returned as ‘empty’.

Only ASCII letters, digits, dashes, underscores and dots are allowed in the final filename. Accents are removed, if possible. Whitespace is replaced by a single dash. Leading or tailing dots or dashes are removed. The filename is limited to 255 characters.

**save**(*destination*, *overwrite=False*, *chunk\_size=65536*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#FileUpload.save)[¶](http://bottle.zzir.cn/index.html#bottle.FileUpload.save)

Save file to disk or copy its content to an open file(-like) object. If *destination* is a directory, [**filename**](http://bottle.zzir.cn/index.html#bottle.FileUpload.filename) is added to the path. Existing files are not overwritten by default (IOError).

|  |  |
| --- | --- |
| **参数:** | * **destination** – File path, directory or file(-like) object. * **overwrite** – If True, replace existing files. (default: False) * **chunk\_size** – Bytes to read at a time. (default: 64kb) |

Exceptions[¶](http://bottle.zzir.cn/index.html#exceptions)

*exception* **BottleException**[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#BottleException)[¶](http://bottle.zzir.cn/index.html#bottle.BottleException)

A base class for exceptions used by bottle.

The [**Bottle**](http://bottle.zzir.cn/index.html#bottle.Bottle) Class[¶](http://bottle.zzir.cn/index.html#the-bottle-class)

*class* **Bottle**(*catchall=True*, *autojson=True*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#Bottle)[¶](http://bottle.zzir.cn/index.html#bottle.Bottle)

Each Bottle object represents a single, distinct web application and consists of routes, callbacks, plugins, resources and configuration. Instances are callable WSGI applications.

|  |  |
| --- | --- |
| **参数:** | **catchall** – If true (default), handle all exceptions. Turn off to let debugging middleware handle exceptions. |

**config** *= None*[¶](http://bottle.zzir.cn/index.html#bottle.Bottle.config)

A [**ConfigDict**](http://bottle.zzir.cn/index.html#bottle.ConfigDict) for app specific configuration.

**resources** *= None*[¶](http://bottle.zzir.cn/index.html#bottle.Bottle.resources)

A [**ResourceManager**](http://bottle.zzir.cn/index.html#bottle.ResourceManager) for application files

**catchall**[¶](http://bottle.zzir.cn/index.html#bottle.Bottle.catchall)

If true, most exceptions are caught and returned as [**HTTPError**](http://bottle.zzir.cn/index.html#bottle.HTTPError)

**add\_hook**(*name*, *func*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#Bottle.add_hook)[¶](http://bottle.zzir.cn/index.html#bottle.Bottle.add_hook)

Attach a callback to a hook. Three hooks are currently implemented:

before\_request

Executed once before each request. The request context is available, but no routing has happened yet.

after\_request

Executed once after each request regardless of its outcome.

app\_reset

Called whenever [**Bottle.reset()**](http://bottle.zzir.cn/index.html#bottle.Bottle.reset) is called.

**remove\_hook**(*name*, *func*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#Bottle.remove_hook)[¶](http://bottle.zzir.cn/index.html#bottle.Bottle.remove_hook)

Remove a callback from a hook.

**trigger\_hook**(*\_Bottle\_\_name*, *\*args*, *\*\*kwargs*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#Bottle.trigger_hook)[¶](http://bottle.zzir.cn/index.html#bottle.Bottle.trigger_hook)

Trigger a hook and return a list of results.

**hook**(*name*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#Bottle.hook)[¶](http://bottle.zzir.cn/index.html#bottle.Bottle.hook)

Return a decorator that attaches a callback to a hook. See [**add\_hook()**](http://bottle.zzir.cn/index.html#bottle.Bottle.add_hook) for details.

**mount**(*prefix*, *app*, *\*\*options*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#Bottle.mount)[¶](http://bottle.zzir.cn/index.html#bottle.Bottle.mount)

Mount an application ([**Bottle**](http://bottle.zzir.cn/index.html#bottle.Bottle) or plain WSGI) to a specific URL prefix. Example:

parent\_app.mount('/prefix/', child\_app)

|  |  |
| --- | --- |
| **参数:** | * **prefix** – path prefix or *mount-point*. * **app** – an instance of [**Bottle**](http://bottle.zzir.cn/index.html#bottle.Bottle) or a WSGI application. |

Plugins from the parent application are not applied to the routes of the mounted child application. If you need plugins in the child application, install them separately.

While it is possible to use path wildcards within the prefix path ([**Bottle**](http://bottle.zzir.cn/index.html#bottle.Bottle) childs only), it is highly discouraged.

The prefix path must end with a slash. If you want to access the root of the child application via */prefix* in addition to */prefix/*, consider adding a route with a 307 redirect to the parent application.

**merge**(*routes*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#Bottle.merge)[¶](http://bottle.zzir.cn/index.html#bottle.Bottle.merge)

Merge the routes of another [**Bottle**](http://bottle.zzir.cn/index.html#bottle.Bottle) application or a list of [**Route**](http://bottle.zzir.cn/index.html#bottle.Route) objects into this application. The routes keep their ‘owner’, meaning that the [**Route.app**](http://bottle.zzir.cn/index.html#bottle.Route.app) attribute is not changed.

**install**(*plugin*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#Bottle.install)[¶](http://bottle.zzir.cn/index.html#bottle.Bottle.install)

Add a plugin to the list of plugins and prepare it for being applied to all routes of this application. A plugin may be a simple decorator or an object that implements the [**Plugin**](http://bottle.zzir.cn/index.html#bottle.Plugin) API.

**uninstall**(*plugin*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#Bottle.uninstall)[¶](http://bottle.zzir.cn/index.html#bottle.Bottle.uninstall)

Uninstall plugins. Pass an instance to remove a specific plugin, a type object to remove all plugins that match that type, a string to remove all plugins with a matching name attribute or True to remove all plugins. Return the list of removed plugins.

**reset**(*route=None*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#Bottle.reset)[¶](http://bottle.zzir.cn/index.html#bottle.Bottle.reset)

Reset all routes (force plugins to be re-applied) and clear all caches. If an ID or route object is given, only that specific route is affected.

**close**()[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#Bottle.close)[¶](http://bottle.zzir.cn/index.html#bottle.Bottle.close)

Close the application and all installed plugins.

**run**(*\*\*kwargs*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#Bottle.run)[¶](http://bottle.zzir.cn/index.html#bottle.Bottle.run)

Calls [**run()**](http://bottle.zzir.cn/index.html#bottle.run) with the same parameters.

**match**(*environ*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#Bottle.match)[¶](http://bottle.zzir.cn/index.html#bottle.Bottle.match)

Search for a matching route and return a ([**Route**](http://bottle.zzir.cn/index.html#bottle.Route) , urlargs) tuple. The second value is a dictionary with parameters extracted from the URL. Raise [**HTTPError**](http://bottle.zzir.cn/index.html#bottle.HTTPError) (404/405) on a non-match.

**get\_url**(*routename*, *\*\*kargs*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#Bottle.get_url)[¶](http://bottle.zzir.cn/index.html#bottle.Bottle.get_url)

Return a string that matches a named route

**add\_route**(*route*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#Bottle.add_route)[¶](http://bottle.zzir.cn/index.html#bottle.Bottle.add_route)

Add a route object, but do not change the [**Route.app**](http://bottle.zzir.cn/index.html#bottle.Route.app) attribute.

**route**(*path=None*, *method='GET'*, *callback=None*, *name=None*, *apply=None*, *skip=None*, *\*\*config*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#Bottle.route)[¶](http://bottle.zzir.cn/index.html#bottle.Bottle.route)

A decorator to bind a function to a request URL. Example:

@app.route('/hello/<**name**>')

def hello(name):

return 'Hello %s' % name

The <name> part is a wildcard. See **Router** for syntax details.

|  |  |
| --- | --- |
| **参数:** | * **path** – Request path or a list of paths to listen to. If no path is specified, it is automatically generated from the signature of the function. * **method** – HTTP method (*GET*, *POST*, *PUT*, ...) or a list of methods to listen to. (default: *GET*) * **callback** – An optional shortcut to avoid the decorator syntax. route(..., callback=func) equals route(...)(func) * **name** – The name for this route. (default: None) * **apply** – A decorator or plugin or a list of plugins. These are applied to the route callback in addition to installed plugins. * **skip** – A list of plugins, plugin classes or names. Matching plugins are not installed to this route. True skips all. |

Any additional keyword arguments are stored as route-specific configuration and passed to plugins (see [**Plugin.apply()**](http://bottle.zzir.cn/index.html#bottle.Plugin.apply)).

**get**(*path=None*, *method='GET'*, *\*\*options*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#Bottle.get)[¶](http://bottle.zzir.cn/index.html#bottle.Bottle.get)

Equals [**route()**](http://bottle.zzir.cn/index.html#bottle.route).

**post**(*path=None*, *method='POST'*, *\*\*options*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#Bottle.post)[¶](http://bottle.zzir.cn/index.html#bottle.Bottle.post)

Equals [**route()**](http://bottle.zzir.cn/index.html#bottle.route) with a POST method parameter.

**put**(*path=None*, *method='PUT'*, *\*\*options*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#Bottle.put)[¶](http://bottle.zzir.cn/index.html#bottle.Bottle.put)

Equals [**route()**](http://bottle.zzir.cn/index.html#bottle.route) with a PUT method parameter.

**delete**(*path=None*, *method='DELETE'*, *\*\*options*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#Bottle.delete)[¶](http://bottle.zzir.cn/index.html#bottle.Bottle.delete)

Equals [**route()**](http://bottle.zzir.cn/index.html#bottle.route) with a DELETE method parameter.

**patch**(*path=None*, *method='PATCH'*, *\*\*options*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#Bottle.patch)[¶](http://bottle.zzir.cn/index.html#bottle.Bottle.patch)

Equals [**route()**](http://bottle.zzir.cn/index.html#bottle.route) with a PATCH method parameter.

**error**(*code=500*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#Bottle.error)[¶](http://bottle.zzir.cn/index.html#bottle.Bottle.error)

Decorator: Register an output handler for a HTTP error code

**wsgi**(*environ*, *start\_response*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#Bottle.wsgi)[¶](http://bottle.zzir.cn/index.html#bottle.Bottle.wsgi)

The bottle WSGI-interface.

*class* **Route**(*app*, *rule*, *method*, *callback*, *name=None*, *plugins=None*, *skiplist=None*, *\*\*config*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#Route)[¶](http://bottle.zzir.cn/index.html#bottle.Route)

This class wraps a route callback along with route specific metadata and configuration and applies Plugins on demand. It is also responsible for turing an URL path rule into a regular expression usable by the Router.

**app** *= None*[¶](http://bottle.zzir.cn/index.html#bottle.Route.app)

The application this route is installed to.

**rule** *= None*[¶](http://bottle.zzir.cn/index.html#bottle.Route.rule)

The path-rule string (e.g. /wiki/<page>).

**method** *= None*[¶](http://bottle.zzir.cn/index.html#bottle.Route.method)

HTTP方法的字符串(例如： GET)

**callback** *= None*[¶](http://bottle.zzir.cn/index.html#bottle.Route.callback)

未应用任何插件的原始回调函数，用于内省。

**name** *= None*[¶](http://bottle.zzir.cn/index.html#bottle.Route.name)

route的名字，如未指定则为 None

**plugins** *= None*[¶](http://bottle.zzir.cn/index.html#bottle.Route.plugins)

A list of route-specific plugins (see [**Bottle.route()**](http://bottle.zzir.cn/index.html#bottle.Bottle.route)).

**skiplist** *= None*[¶](http://bottle.zzir.cn/index.html#bottle.Route.skiplist)

A list of plugins to not apply to this route (see [**Bottle.route()**](http://bottle.zzir.cn/index.html#bottle.Bottle.route)).

**config** *= None*[¶](http://bottle.zzir.cn/index.html#bottle.Route.config)

Additional keyword arguments passed to the [**Bottle.route()**](http://bottle.zzir.cn/index.html#bottle.Bottle.route) decorator are stored in this dictionary. Used for route-specific plugin configuration and meta-data.

**call**[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#Route.call)[¶](http://bottle.zzir.cn/index.html#bottle.Route.call)

The route callback with all plugins applied. This property is created on demand and then cached to speed up subsequent requests.

**reset**()[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#Route.reset)[¶](http://bottle.zzir.cn/index.html#bottle.Route.reset)

Forget any cached values. The next time [**call**](http://bottle.zzir.cn/index.html#bottle.Route.call) is accessed, all plugins are re-applied.

**prepare**()[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#Route.prepare)[¶](http://bottle.zzir.cn/index.html#bottle.Route.prepare)

Do all on-demand work immediately (useful for debugging).

**all\_plugins**()[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#Route.all_plugins)[¶](http://bottle.zzir.cn/index.html#bottle.Route.all_plugins)

Yield all Plugins affecting this route.

**get\_undecorated\_callback**()[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#Route.get_undecorated_callback)[¶](http://bottle.zzir.cn/index.html#bottle.Route.get_undecorated_callback)

Return the callback. If the callback is a decorated function, try to recover the original function.

**get\_callback\_args**()[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#Route.get_callback_args)[¶](http://bottle.zzir.cn/index.html#bottle.Route.get_callback_args)

Return a list of argument names the callback (most likely) accepts as keyword arguments. If the callback is a decorated function, try to recover the original function before inspection.

**get\_config**(*key*, *default=None*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#Route.get_config)[¶](http://bottle.zzir.cn/index.html#bottle.Route.get_config)

Lookup a config field and return its value, first checking the route.config, then route.app.config.

The [**Request**](http://bottle.zzir.cn/index.html#bottle.Request) Object[¶](http://bottle.zzir.cn/index.html#the-request-object)

The [**Request**](http://bottle.zzir.cn/index.html#bottle.Request) class wraps a WSGI environment and provides helpful methods to parse and access form data, cookies, file uploads and other metadata. Most of the attributes are read-only.

**Request**[¶](http://bottle.zzir.cn/index.html#bottle.Request)

[**BaseRequest**](http://bottle.zzir.cn/index.html#bottle.BaseRequest) 的别名

*class* **BaseRequest**(*environ=None*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#BaseRequest)[¶](http://bottle.zzir.cn/index.html#bottle.BaseRequest)

A wrapper for WSGI environment dictionaries that adds a lot of convenient access methods and properties. Most of them are read-only.

Adding new attributes to a request actually adds them to the environ dictionary (as ‘bottle.request.ext.<name>’). This is the recommended way to store and access request-specific data.

**MEMFILE\_MAX** *= 102400*[¶](http://bottle.zzir.cn/index.html#bottle.BaseRequest.MEMFILE_MAX)

Maximum size of memory buffer for [**body**](http://bottle.zzir.cn/index.html#bottle.BaseRequest.body) in bytes.

**environ**[¶](http://bottle.zzir.cn/index.html#bottle.BaseRequest.environ)

The wrapped WSGI environ dictionary. This is the only real attribute. All other attributes actually are read-only properties.

**app**[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#BaseRequest.app)[¶](http://bottle.zzir.cn/index.html#bottle.BaseRequest.app)

Bottle application handling this request.

**route**[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#BaseRequest.route)[¶](http://bottle.zzir.cn/index.html#bottle.BaseRequest.route)

The bottle [**Route**](http://bottle.zzir.cn/index.html#bottle.Route) object that matches this request.

**url\_args**[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#BaseRequest.url_args)[¶](http://bottle.zzir.cn/index.html#bottle.BaseRequest.url_args)

The arguments extracted from the URL.

**path**[¶](http://bottle.zzir.cn/index.html#bottle.BaseRequest.path)

The value of PATH\_INFO with exactly one prefixed slash (to fix broken clients and avoid the “empty path” edge case).

**method**[¶](http://bottle.zzir.cn/index.html#bottle.BaseRequest.method)

The REQUEST\_METHOD value as an uppercase string.

**headers**[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#BaseRequest.headers)[¶](http://bottle.zzir.cn/index.html#bottle.BaseRequest.headers)

A [**WSGIHeaderDict**](http://bottle.zzir.cn/index.html#bottle.WSGIHeaderDict) that provides case-insensitive access to HTTP request headers.

**get\_header**(*name*, *default=None*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#BaseRequest.get_header)[¶](http://bottle.zzir.cn/index.html#bottle.BaseRequest.get_header)

Return the value of a request header, or a given default value.

**cookies**[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#BaseRequest.cookies)[¶](http://bottle.zzir.cn/index.html#bottle.BaseRequest.cookies)

Cookies parsed into a [**FormsDict**](http://bottle.zzir.cn/index.html#bottle.FormsDict). Signed cookies are NOT decoded. Use [**get\_cookie()**](http://bottle.zzir.cn/index.html#bottle.BaseRequest.get_cookie) if you expect signed cookies.

**get\_cookie**(*key*, *default=None*, *secret=None*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#BaseRequest.get_cookie)[¶](http://bottle.zzir.cn/index.html#bottle.BaseRequest.get_cookie)

Return the content of a cookie. To read a *Signed Cookie*, the *secret* must match the one used to create the cookie (see [**BaseResponse.set\_cookie()**](http://bottle.zzir.cn/index.html#bottle.BaseResponse.set_cookie)). If anything goes wrong (missing cookie or wrong signature), return a default value.

**query**[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#BaseRequest.query)[¶](http://bottle.zzir.cn/index.html#bottle.BaseRequest.query)

The [**query\_string**](http://bottle.zzir.cn/index.html#bottle.BaseRequest.query_string) parsed into a [**FormsDict**](http://bottle.zzir.cn/index.html#bottle.FormsDict). These values are sometimes called “URL arguments” or “GET parameters”, but not to be confused with “URL wildcards” as they are provided by the **Router**.

**forms**[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#BaseRequest.forms)[¶](http://bottle.zzir.cn/index.html#bottle.BaseRequest.forms)

Form values parsed from an *url-encoded* or *multipart/form-data* encoded POST or PUT request body. The result is returned as a [**FormsDict**](http://bottle.zzir.cn/index.html#bottle.FormsDict). All keys and values are strings. File uploads are stored separately in [**files**](http://bottle.zzir.cn/index.html#bottle.BaseRequest.files).

**params**[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#BaseRequest.params)[¶](http://bottle.zzir.cn/index.html#bottle.BaseRequest.params)

A [**FormsDict**](http://bottle.zzir.cn/index.html#bottle.FormsDict) with the combined values of [**query**](http://bottle.zzir.cn/index.html#bottle.BaseRequest.query) and [**forms**](http://bottle.zzir.cn/index.html#bottle.BaseRequest.forms). File uploads are stored in [**files**](http://bottle.zzir.cn/index.html#bottle.BaseRequest.files).

**files**[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#BaseRequest.files)[¶](http://bottle.zzir.cn/index.html#bottle.BaseRequest.files)

File uploads parsed from *multipart/form-data* encoded POST or PUT request body. The values are instances of [**FileUpload**](http://bottle.zzir.cn/index.html#bottle.FileUpload).

**json**[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#BaseRequest.json)[¶](http://bottle.zzir.cn/index.html#bottle.BaseRequest.json)

If the Content-Type header is application/json or application/json-rpc, this property holds the parsed content of the request body. Only requests smaller than [**MEMFILE\_MAX**](http://bottle.zzir.cn/index.html#bottle.BaseRequest.MEMFILE_MAX) are processed to avoid memory exhaustion. Invalid JSON raises a 400 error response.

**body**[¶](http://bottle.zzir.cn/index.html#bottle.BaseRequest.body)

The HTTP request body as a seek-able file-like object. Depending on [**MEMFILE\_MAX**](http://bottle.zzir.cn/index.html#bottle.BaseRequest.MEMFILE_MAX), this is either a temporary file or a **io.BytesIO** instance. Accessing this property for the first time reads and replaces the wsgi.input environ variable. Subsequent accesses just do a *seek(0)* on the file object.

**chunked**[¶](http://bottle.zzir.cn/index.html#bottle.BaseRequest.chunked)

True if Chunked transfer encoding was.

**GET**[¶](http://bottle.zzir.cn/index.html#bottle.BaseRequest.GET)

An alias for [**query**](http://bottle.zzir.cn/index.html#bottle.BaseRequest.query).

**POST**[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#BaseRequest.POST)[¶](http://bottle.zzir.cn/index.html#bottle.BaseRequest.POST)

The values of [**forms**](http://bottle.zzir.cn/index.html#bottle.BaseRequest.forms) and [**files**](http://bottle.zzir.cn/index.html#bottle.BaseRequest.files) combined into a single [**FormsDict**](http://bottle.zzir.cn/index.html#bottle.FormsDict). Values are either strings (form values) or instances of **cgi.FieldStorage** (file uploads).

**url**[¶](http://bottle.zzir.cn/index.html#bottle.BaseRequest.url)

The full request URI including hostname and scheme. If your app lives behind a reverse proxy or load balancer and you get confusing results, make sure that the X-Forwarded-Host header is set correctly.

**urlparts**[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#BaseRequest.urlparts)[¶](http://bottle.zzir.cn/index.html#bottle.BaseRequest.urlparts)

The [**url**](http://bottle.zzir.cn/index.html#bottle.BaseRequest.url) string as an **urlparse.SplitResult** tuple. The tuple contains (scheme, host, path, query\_string and fragment), but the fragment is always empty because it is not visible to the server.

**fullpath**[¶](http://bottle.zzir.cn/index.html#bottle.BaseRequest.fullpath)

Request path including [**script\_name**](http://bottle.zzir.cn/index.html#bottle.BaseRequest.script_name) (if present).

**query\_string**[¶](http://bottle.zzir.cn/index.html#bottle.BaseRequest.query_string)

The raw [**query**](http://bottle.zzir.cn/index.html#bottle.BaseRequest.query) part of the URL (everything in between ? and #) as a string.

**script\_name**[¶](http://bottle.zzir.cn/index.html#bottle.BaseRequest.script_name)

The initial portion of the URL’s *path* that was removed by a higher level (server or routing middleware) before the application was called. This script path is returned with leading and tailing slashes.

**path\_shift**(*shift=1*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#BaseRequest.path_shift)[¶](http://bottle.zzir.cn/index.html#bottle.BaseRequest.path_shift)

Shift path segments from [**path**](http://bottle.zzir.cn/index.html#bottle.BaseRequest.path) to [**script\_name**](http://bottle.zzir.cn/index.html#bottle.BaseRequest.script_name) and

vice versa.

|  |  |
| --- | --- |
| **参数:** | **shift** – The number of path segments to shift. May be negative to change the shift direction. (default: 1) |

**content\_length**[¶](http://bottle.zzir.cn/index.html#bottle.BaseRequest.content_length)

The request body length as an integer. The client is responsible to set this header. Otherwise, the real length of the body is unknown and -1 is returned. In this case, [**body**](http://bottle.zzir.cn/index.html#bottle.BaseRequest.body) will be empty.

**content\_type**[¶](http://bottle.zzir.cn/index.html#bottle.BaseRequest.content_type)

The Content-Type header as a lowercase-string (default: empty).

**is\_xhr**[¶](http://bottle.zzir.cn/index.html#bottle.BaseRequest.is_xhr)

True if the request was triggered by a XMLHttpRequest. This only works with JavaScript libraries that support the *X-Requested-With* header (most of the popular libraries do).

**is\_ajax**[¶](http://bottle.zzir.cn/index.html#bottle.BaseRequest.is_ajax)

Alias for [**is\_xhr**](http://bottle.zzir.cn/index.html#bottle.BaseRequest.is_xhr). “Ajax” is not the right term.

**auth**[¶](http://bottle.zzir.cn/index.html#bottle.BaseRequest.auth)

HTTP authentication data as a (user, password) tuple. This implementation currently supports basic (not digest) authentication only. If the authentication happened at a higher level (e.g. in the front web-server or a middleware), the password field is None, but the user field is looked up from the REMOTE\_USER environ variable. On any errors, None is returned.

**remote\_route**[¶](http://bottle.zzir.cn/index.html#bottle.BaseRequest.remote_route)

A list of all IPs that were involved in this request, starting with the client IP and followed by zero or more proxies. This does only work if all proxies support the X-Forwarded-For header. Note that this information can be forged by malicious clients.

**remote\_addr**[¶](http://bottle.zzir.cn/index.html#bottle.BaseRequest.remote_addr)

The client IP as a string. Note that this information can be forged by malicious clients.

**copy**()[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#BaseRequest.copy)[¶](http://bottle.zzir.cn/index.html#bottle.BaseRequest.copy)

Return a new [**Request**](http://bottle.zzir.cn/index.html#bottle.Request) with a shallow [**environ**](http://bottle.zzir.cn/index.html#bottle.BaseRequest.environ) copy.

The module-level [**bottle.request**](http://bottle.zzir.cn/index.html#bottle.request) is a proxy object (implemented in [**LocalRequest**](http://bottle.zzir.cn/index.html#bottle.LocalRequest)) and always refers to the *current* request, or in other words, the request that is currently processed by the request handler in the current thread. This *thread locality* ensures that you can safely use a global instance in a multi-threaded environment.

*class* **LocalRequest**(*environ=None*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#LocalRequest)[¶](http://bottle.zzir.cn/index.html#bottle.LocalRequest)

A thread-local subclass of [**BaseRequest**](http://bottle.zzir.cn/index.html#bottle.BaseRequest) with a different set of attributes for each thread. There is usually only one global instance of this class ([**request**](http://bottle.zzir.cn/index.html#bottle.request)). If accessed during a request/response cycle, this instance always refers to the *current* request (even on a multithreaded server).

**bind**(*environ=None*)[¶](http://bottle.zzir.cn/index.html#bottle.LocalRequest.bind)

Wrap a WSGI environ dictionary.

**environ**[¶](http://bottle.zzir.cn/index.html#bottle.LocalRequest.environ)

Thread-local property

**request** *= <LocalRequest: GET http://127.0.0.1/>*

A thread-safe instance of [**LocalRequest**](http://bottle.zzir.cn/index.html#bottle.LocalRequest). If accessed from within a request callback, this instance always refers to the *current* request (even on a multi-threaded server).

[**Response**](http://bottle.zzir.cn/index.html#bottle.Response) 对象[¶](http://bottle.zzir.cn/index.html#the-response-object)

The [**Response**](http://bottle.zzir.cn/index.html#bottle.Response) class stores the HTTP status code as well as headers and cookies that are to be sent to the client. Similar to [**bottle.request**](http://bottle.zzir.cn/index.html#bottle.request) there is a thread-local [**bottle.response**](http://bottle.zzir.cn/index.html#bottle.response) instance that can be used to adjust the *current* response. Moreover, you can instantiate [**Response**](http://bottle.zzir.cn/index.html#bottle.Response) and return it from your request handler. In this case, the custom instance overrules the headers and cookies defined in the global one.

**Response**[¶](http://bottle.zzir.cn/index.html#bottle.Response)

[**BaseResponse**](http://bottle.zzir.cn/index.html#bottle.BaseResponse) 的别名

*class* **BaseResponse**(*body=''*, *status=None*, *headers=None*, *\*\*more\_headers*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#BaseResponse)[¶](http://bottle.zzir.cn/index.html#bottle.BaseResponse)

Storage class for a response body as well as headers and cookies.

This class does support dict-like case-insensitive item-access to headers, but is NOT a dict. Most notably, iterating over a response yields parts of the body and not the headers.

|  |  |
| --- | --- |
| **参数:** | * **body** – The response body as one of the supported types. * **status** – Either an HTTP status code (e.g. 200) or a status line including the reason phrase (e.g. ‘200 OK’). * **headers** – A dictionary or a list of name-value pairs. |

Additional keyword arguments are added to the list of headers. Underscores in the header name are replaced with dashes.

**copy**(*cls=None*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#BaseResponse.copy)[¶](http://bottle.zzir.cn/index.html#bottle.BaseResponse.copy)

Returns a copy of self.

**status\_line**[¶](http://bottle.zzir.cn/index.html#bottle.BaseResponse.status_line)

The HTTP status line as a string (e.g. 404 Not Found).

**status\_code**[¶](http://bottle.zzir.cn/index.html#bottle.BaseResponse.status_code)

The HTTP status code as an integer (e.g. 404).

**status**[¶](http://bottle.zzir.cn/index.html#bottle.BaseResponse.status)

A writeable property to change the HTTP response status. It accepts either a numeric code (100-999) or a string with a custom reason phrase (e.g. “404 Brain not found”). Both [**status\_line**](http://bottle.zzir.cn/index.html#bottle.BaseResponse.status_line) and [**status\_code**](http://bottle.zzir.cn/index.html#bottle.BaseResponse.status_code) are updated accordingly. The return value is always a status string.

**headers**[¶](http://bottle.zzir.cn/index.html#bottle.BaseResponse.headers)

An instance of [**HeaderDict**](http://bottle.zzir.cn/index.html#bottle.HeaderDict), a case-insensitive dict-like view on the response headers.

**get\_header**(*name*, *default=None*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#BaseResponse.get_header)[¶](http://bottle.zzir.cn/index.html#bottle.BaseResponse.get_header)

Return the value of a previously defined header. If there is no header with that name, return a default value.

**set\_header**(*name*, *value*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#BaseResponse.set_header)[¶](http://bottle.zzir.cn/index.html#bottle.BaseResponse.set_header)

Create a new response header, replacing any previously defined headers with the same name.

**add\_header**(*name*, *value*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#BaseResponse.add_header)[¶](http://bottle.zzir.cn/index.html#bottle.BaseResponse.add_header)

Add an additional response header, not removing duplicates.

**iter\_headers**()[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#BaseResponse.iter_headers)[¶](http://bottle.zzir.cn/index.html#bottle.BaseResponse.iter_headers)

Yield (header, value) tuples, skipping headers that are not allowed with the current response status code.

**headerlist**[¶](http://bottle.zzir.cn/index.html#bottle.BaseResponse.headerlist)

WSGI conform list of (header, value) tuples.

**content\_type**[¶](http://bottle.zzir.cn/index.html#bottle.BaseResponse.content_type)

Current value of the ‘Content-Type’ header.

**content\_length**[¶](http://bottle.zzir.cn/index.html#bottle.BaseResponse.content_length)

Current value of the ‘Content-Length’ header.

**expires**[¶](http://bottle.zzir.cn/index.html#bottle.BaseResponse.expires)

Current value of the ‘Expires’ header.

**charset**[¶](http://bottle.zzir.cn/index.html#bottle.BaseResponse.charset)

Return the charset specified in the content-type header (default: utf8).

**set\_cookie**(*name*, *value*, *secret=None*, *\*\*options*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#BaseResponse.set_cookie)[¶](http://bottle.zzir.cn/index.html#bottle.BaseResponse.set_cookie)

Create a new cookie or replace an old one. If the *secret* parameter is set, create a *Signed Cookie* (described below).

|  |  |
| --- | --- |
| **参数:** | * **name** – the name of the cookie. * **value** – the value of the cookie. * **secret** – a signature key required for signed cookies. |

Additionally, this method accepts all RFC 2109 attributes that are supported by **cookie.Morsel**, including:

|  |  |
| --- | --- |
| **参数:** | * **max\_age** – maximum age in seconds. (default: None) * **expires** – a datetime object or UNIX timestamp. (default: None) * **domain** – the domain that is allowed to read the cookie. (default: current domain) * **path** – limits the cookie to a given path (default: current path) * **secure** – limit the cookie to HTTPS connections (default: off). * **httponly** – prevents client-side javascript to read this cookie (default: off, requires Python 2.6 or newer). |

If neither *expires* nor *max\_age* is set (default), the cookie will expire at the end of the browser session (as soon as the browser window is closed).

Signed cookies may store any pickle-able object and are cryptographically signed to prevent manipulation. Keep in mind that cookies are limited to 4kb in most browsers.

Warning: Signed cookies are not encrypted (the client can still see the content) and not copy-protected (the client can restore an old cookie). The main intention is to make pickling and unpickling save, not to store secret information at client side.

**delete\_cookie**(*key*, *\*\*kwargs*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#BaseResponse.delete_cookie)[¶](http://bottle.zzir.cn/index.html#bottle.BaseResponse.delete_cookie)

Delete a cookie. Be sure to use the same *domain* and *path* settings as used to create the cookie.

*class* **LocalResponse**(*body=''*, *status=None*, *headers=None*, *\*\*more\_headers*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#LocalResponse)[¶](http://bottle.zzir.cn/index.html#bottle.LocalResponse)

A thread-local subclass of [**BaseResponse**](http://bottle.zzir.cn/index.html#bottle.BaseResponse) with a different set of attributes for each thread. There is usually only one global instance of this class ([**response**](http://bottle.zzir.cn/index.html#bottle.response)). Its attributes are used to build the HTTP response at the end of the request/response cycle.

**body**[¶](http://bottle.zzir.cn/index.html#bottle.LocalResponse.body)

Thread-local property

The following two classes can be raised as an exception. The most noticeable difference is that bottle invokes error handlers for [**HTTPError**](http://bottle.zzir.cn/index.html#bottle.HTTPError), but not for [**HTTPResponse**](http://bottle.zzir.cn/index.html#bottle.HTTPResponse) or other response types.

*exception* **HTTPResponse**(*body=''*, *status=None*, *headers=None*, *\*\*more\_headers*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#HTTPResponse)[¶](http://bottle.zzir.cn/index.html#bottle.HTTPResponse)

*exception* **HTTPError**(*status=None*, *body=None*, *exception=None*, *traceback=None*, *\*\*more\_headers*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#HTTPError)[¶](http://bottle.zzir.cn/index.html#bottle.HTTPError)

模板[¶](http://bottle.zzir.cn/index.html#templates)

All template engines supported by [**bottle**](http://bottle.zzir.cn/index.html#module-bottle) implement the [**BaseTemplate**](http://bottle.zzir.cn/index.html#bottle.BaseTemplate) API. This way it is possible to switch and mix template engines without changing the application code at all.

*class* **BaseTemplate**(*source=None*, *name=None*, *lookup=None*, *encoding='utf8'*, *\*\*settings*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#BaseTemplate)[¶](http://bottle.zzir.cn/index.html#bottle.BaseTemplate)

Base class and minimal API for template adapters

**\_\_init\_\_**(*source=None*, *name=None*, *lookup=None*, *encoding='utf8'*, *\*\*settings*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#BaseTemplate.__init__)[¶](http://bottle.zzir.cn/index.html#bottle.BaseTemplate.__init__)

Create a new template. If the source parameter (str or buffer) is missing, the name argument is used to guess a template filename. Subclasses can assume that self.source and/or self.filename are set. Both are strings. The lookup, encoding and settings parameters are stored as instance variables. The lookup parameter stores a list containing directory paths. The encoding parameter should be used to decode byte strings or files. The settings parameter contains a dict for engine-specific settings.

*classmethod* **search**(*name*, *lookup=None*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#BaseTemplate.search)[¶](http://bottle.zzir.cn/index.html#bottle.BaseTemplate.search)

Search name in all directories specified in lookup. First without, then with common extensions. Return first hit.

*classmethod* **global\_config**(*key*, *\*args*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#BaseTemplate.global_config)[¶](http://bottle.zzir.cn/index.html#bottle.BaseTemplate.global_config)

This reads or sets the global settings stored in class.settings.

**prepare**(*\*\*options*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#BaseTemplate.prepare)[¶](http://bottle.zzir.cn/index.html#bottle.BaseTemplate.prepare)

Run preparations (parsing, caching, ...). It should be possible to call this again to refresh a template or to update settings.

**render**(*\*args*, *\*\*kwargs*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#BaseTemplate.render)[¶](http://bottle.zzir.cn/index.html#bottle.BaseTemplate.render)

Render the template with the specified local variables and return a single byte or unicode string. If it is a byte string, the encoding must match self.encoding. This method must be thread-safe! Local variables may be provided in dictionaries (args) or directly, as keywords (kwargs).

**view**(*tpl\_name*, *\*\*defaults*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#view)[¶](http://bottle.zzir.cn/index.html#bottle.view)

Decorator: renders a template for a handler. The handler can control its behavior like that:

* return a dict of template vars to fill out the template
* return something other than a dict and the view decorator will not process the template, but return the handler result as is. This includes returning a HTTPResponse(dict) to get, for instance, JSON with autojson or other castfilters.

**template**(*\*args*, *\*\*kwargs*)[[源代码]](http://bottle.zzir.cn/_modules/bottle.html#template)[¶](http://bottle.zzir.cn/index.html#bottle.template)

Get a rendered template as a string iterator. You can use a name, a filename or a template string as first parameter. Template rendering arguments can be passed as dictionaries or directly (as keyword arguments).

You can write your own adapter for your favourite template engine or use one of the predefined adapters. Currently there are four fully supported template engines:

| **Class** | **URL** | **Decorator** | **Render function** |
| --- | --- | --- | --- |
| [**SimpleTemplate**](http://bottle.zzir.cn/index.html#bottle.SimpleTemplate) | [SimpleTemplate 模板引擎](http://bottle.zzir.cn/index.html#document-stpl) | [**view()**](http://bottle.zzir.cn/index.html#bottle.view) | [**template()**](http://bottle.zzir.cn/index.html#bottle.template) |
| **MakoTemplate** | [http://www.makotemplates.org](http://www.makotemplates.org/) | **mako\_view()** | **mako\_template()** |
| **CheetahTemplate** | <http://www.cheetahtemplate.org/> | **cheetah\_view()** | **cheetah\_template()** |
| **Jinja2Template** | <http://jinja.pocoo.org/> | **jinja2\_view()** | **jinja2\_template()** |

To use **MakoTemplate** as your default template engine, just import its specialised decorator and render function:

from bottle import mako\_view as view, mako\_template as template

可用插件列表[¶](http://bottle.zzir.cn/index.html#list-of-available-plugins)

这是一份第三方插件的列表，扩展Bottle的核心功能，或集成其它类库。

在 [插件](http://bottle.zzir.cn/index.html#plugins) 查看常见的插件问题(安装，使用)。如果你计划开发一个新的插件， [插件开发指南](http://bottle.zzir.cn/index.html#document-plugindev) 也许对你有帮助。

[Bottle-Beaker](http://pypi.python.org/pypi/bottle-beaker/)

Beaker to session and caching library with WSGI Middleware

[Bottle-Cork](http://cork.firelet.net/)

Cork插件基于Bottle，提供了一些简单的方法来实现Web应用的权限验证。

[Bottle-Extras](http://pypi.python.org/pypi/bottle-extras/)

安装Bottle插件的集合

[Bottle-Flash](http://pypi.python.org/pypi/bottle-flash/)

Bottle的flash插件

[Bottle-Hotqueue](http://pypi.python.org/pypi/bottle-hotqueue/)

基于redis的FIFO队列服务

[Macaron](http://nobrin.github.com/macaron/webapp.html)

Macaron是用于SQLite的ORM

[Bottle-Memcache](http://pypi.python.org/pypi/bottle-memcache/)

Memcache集成

[Bottle-Mongo](http://pypi.python.org/pypi/bottle-mongo/)

MongoDB集成

[Bottle-Redis](http://pypi.python.org/pypi/bottle-redis/)

Redis集成

[Bottle-Renderer](http://pypi.python.org/pypi/bottle-renderer/)

Renderer插件

[Bottle-Servefiles](http://pypi.python.org/pypi/bottle-servefiles/)

一个可重用的APP，为Bottle应用提供静态文件服务。

[Bottle-Sqlalchemy](http://pypi.python.org/pypi/bottle-sqlalchemy/)

SQLAlchemy集成

[Bottle-Sqlite](http://pypi.python.org/pypi/bottle-sqlite/)

SQLite3数据库集成

[Bottle-Web2pydal](http://pypi.python.org/pypi/bottle-web2pydal/)

Wbe2py的Dal集成

[Bottle-Werkzeug](http://pypi.python.org/pypi/bottle-werkzeug/)

集成 *werkzeug* （可选的request和response对象，更高级的调试中间件等等）

这里列出的插件不属于Bottle或Bottle项目，是第三方开发并维护的。

知识库[¶](http://bottle.zzir.cn/index.html#knowledge-base)

收集文章，使用指南和HOWTO

[指南](http://bottle.zzir.cn/index.html#id1)[¶](http://bottle.zzir.cn/index.html#tutorial-todo-list-application)

注解

这份教程是 [noisefloor](http://github.com/noisefloor) 编写的，并在不断完善中。

这份教程简单介绍了Bottle框架，目的是让你看完后能在项目中使用Bottle。它没有涵盖所有东西，但介绍了URL映射，模板，处理GET/POST请求等基础知识。

读懂这份教程，你不需要事先了解WSGI标准，Bottle也一直避免用户直接接触WSGI标准。但你需要了解 [Python](http://www.python.org/) 这门语言。更进一步，这份教程中的例子需要从SQL数据库中读写数据，所以事先了解一点SQL知识是很有帮助的。例子中使用了 [SQLite](http://www.sqlite.org/) 来保存数据。因为是网页应用，所以事先了解一点HTML的知识也很有帮助。

作为一份教程，我们的代码尽可能做到了简明扼要。尽管教程中的代码能够工作，但是我们还是不建议你在公共服务器中使用教程中的代码。如果你想要这样做，你应该添加足够的错误处理，并且加密你的数据库，处理用户的输入。

**Table of Contents**

* [指南](http://bottle.zzir.cn/index.html#tutorial-todo-list-application)
  + [目标](http://bottle.zzir.cn/index.html#goals)
  + [开始之前...](http://bottle.zzir.cn/index.html#before-we-start)
  + [基于Bottle的待办事项列表](http://bottle.zzir.cn/index.html#using-bottle-for-a-web-based-todo-list)
  + [安装服务器](http://bottle.zzir.cn/index.html#server-setup)
  + [结语](http://bottle.zzir.cn/index.html#final-words)
  + [完整代码](http://bottle.zzir.cn/index.html#complete-example-listing)

[目标](http://bottle.zzir.cn/index.html#id2)[¶](http://bottle.zzir.cn/index.html#goals)

在这份教程结束的时候，我们将完成一个简单的，基于Web的ToDo list(待办事项列表)。列表中的每一个待办事项都包含一条文本(最长100个字符)和一个状态(0表示关闭，1表示开启)。通过网页，已开启的待办事项可以被查看和编辑，可添加待办事项到列表中。

在开发过程中，所有的页面都只可以通过 localhost 来访问，完了会介绍如何将应用部署到”真实”服务器的服务器上面，包括使用mod\_wsgi来部署到Apache服务器上面。

Bottle会负责URL映射，通过模板来输出页面。待办事项列表被存储在一个SQLite数据库中，通过Python代码来读写数据库。

我们会完成以下页面和功能:

* 首页 http://localhost:8080/todo
* 添加待办事项: http://localhost:8080/new
* page for editing items: http://localhost:8080/edit/<no:int>
* 捕获错误

[开始之前...](http://bottle.zzir.cn/index.html#id3)[¶](http://bottle.zzir.cn/index.html#before-we-start)

**安装Bottle**

假设你已经安装好了Python (2.5或更改版本)，接下来你只需要下载Bottle就行了。除了Python标准库，Bottle没有其他依赖。

你可通过Python的esay\_install命令来安装Bottle: easy\_install bottle

**其它软件**

因为我们使用SQLite3来做数据库，请确保它已安装。如果是Linux系统，大多数的发行版已经默认安装了SQLite3。SQLite同时可工作在Windows系统和MacOS X系统上面。Pyhton标准库中，已经包含了 *sqlite3* 模块。

**创建一个SQL数据库**

首先，我们需要先创建一个数据库，稍后会用到。在你的项目文件夹执行以下脚本即可，你也可以在Python解释器逐条执行。

import sqlite3

con = sqlite3.connect('todo.db') # Warning: This file is created in the current directory

con.execute("CREATE TABLE todo (id INTEGER PRIMARY KEY, task char(100) NOT NULL, status bool NOT NULL)")

con.execute("INSERT INTO todo (task,status) VALUES ('Read A-byte-of-python to get a good introduction into Python',0)")

con.execute("INSERT INTO todo (task,status) VALUES ('Visit the Python website',1)")

con.execute("INSERT INTO todo (task,status) VALUES ('Test various editors for and check the syntax highlighting',1)")

con.execute("INSERT INTO todo (task,status) VALUES ('Choose your favorite WSGI-Framework',0)")

con.commit()

现在，我们已经创建了一个名字为 *todo.db* 的数据库文件，数据库中有一张名为 todo 的表，表中有 id , task , 及 status 这三列。每一行的 id 都是唯一的，稍后会根据id来获取数据。 task 用于保存待办事项的文本，最大长度为100个字符。最后 status 用于标明待办事项的状态，0为开启，1为关闭。

[基于Bottle的待办事项列表](http://bottle.zzir.cn/index.html#id4)[¶](http://bottle.zzir.cn/index.html#using-bottle-for-a-web-based-todo-list)

为了创建我们的Web应用，我们先来介绍一下Bottle框架。首先，我们需要了解Bottle中的route，即URL映射。

**route URL映射**

基本上，浏览器访问的每一页面都是动态生成的。Bottle通过route，将浏览器访问的URL映射到具体的Python函数。例如，在我们访问 http://localhost:8080/todo 的时候，Bottle会查找 todo 这个route映射到了哪个函数上面，接着调用该函数来响应浏览器请求。

**第一步 - 显示所有已开启的待办事项**

在我们了解什么是route后，让我们来试着写一个。访问它即可查看所有已开启的待办事项

import sqlite3

from bottle import route, run

@route('/todo')

def todo\_list():

conn = sqlite3.connect('todo.db')

c = conn.cursor()

c.execute("SELECT id, task FROM todo WHERE status LIKE '1'")

result = c.fetchall()

return str(result)

run()

将上面的代码保存为 todo.py ，放到 todo.db 文件所在的目录。如果你想将它们分开放，则需要在 sqlite3.connect() 函数中写上 todo.db 文件的路径。

来看看我们写的代码。导入了必须的 sqlite3 模块，从Bottle中导入 route 和 run 。run() 函数启动了Bottle的内置开发服务器，默认情况下，开发服务器在监听本地的8080端口。route 是Bottle实现URL映射功能的修饰器。你可以看到，我们定义了一个 todo\_list() 函数，读取了数据库中的数据。然后我们使用 @route('/todo') 来将 todo\_list() 函数和 todo 这个route绑定在一起。每一次浏览器访问 http://localhost:8080/todo 的时候，Bottle都会调用 todo\_list() 函数来响应请求，并返回页面，这就是route的工作方式了。

事实上，你可以给一个函数添加多个route。

@route('/todo')

@route('/my\_todo\_list')

def todo\_list():

...

这样是正确的。但是反过来，你不能将一个route和多个函数绑定在一起。

你在浏览器中看到的即是你在 todo\_list() 函数中返回的页面。在这个例子中，我们通过 str() 函数将结果转换成字符串，因为Bottle期望函数的返回值是一个字符串或一个字符串的列表。但 [Python DB API](http://www.python.org/dev/peps/pep-0249/) 中规定了，数据库查询的返回值是一个元组的列表。

现在，我们已经了解上面的代码是如何工作的，是时候运行它来看看效果了。记得在Linux或Unix系统中， todo.py 文件需要标记为可执行(译者注：没有必要)。然后，通过 python todo.py 命令来执行该脚本，接着用浏览器访问 http://localhost:8080/todo 来看看效果。如果代码没有写错，你应该会在页面看到以下输出

[(2, u'Visit the Python website'), (3, u'Test various editors for and check the syntax highlighting')]

如果是这样，那么恭喜你！如果出现错误，那么你需要检查代码时候写错，修改完后记得重启HTTP服务器，要不新的版本不会生效。

实际上，这个输出很难看，只是SQL查询的结果。

所以，下一步我们会把它变得更好看。

**调试和自动加载**

或许你已经注意到了，如果代码出错的话，Bottle会在页面上显示一个简短的错误信息。例如，连接数据库失败。为了方便调试， 我们希望错误信息更加具体，可加上以下语句。

from bottle import run, route, debug

...

#add this at the very end:

debug(True)

run()

开启调试模式后，出错时页面会打印出完整的Python运行栈。另外，在调试模式下，模板也不会被缓存，任何对模板的修改会马上生效，而不用重启服务器。

警告

debug(True) 是为开发时的调试服务的， *不应* 在生产环境中开启调试模式。

另外一个十分有用的功能是自动加载，可修改 run() 语句来开启。

run(reloader=True)

这样会自动检测对脚本的修改，并自动重启服务器来使其生效。

同上，这个功能并不建议在生产环境中使用。

**使用模板来格式化输出**

现在我们试着格式化脚本的输出，使其更适合查看。

实际上，Bottle期望route的回调函数返回一个字符串或一个字符串列表，通过内置的HTTP服务器将其返回给浏览器。Bottle不关心字符串的内容，所以我们可以将其格式化成HTML格式。

Bottle内置了独创的模板引擎。模板是后缀名为 .tpl 的文本文件。模板的内容混合着HTML标签和Python语句，模板也可以接受参数。例如数据库的查询结果，我们可以在模板内将其漂亮地格式化。

接下来，我们要将数据库的查询结果格式化为一个两列的表格。表格的第一列为待办事项的ID，第二列为待办事项的内容。查询结果是一个元组的列表，列表中的每个元组后包含一个结果。

在例子中使用模板，只需要添加以下代码。

from bottle import route, run, debug, template

...

result = c.fetchall()

c.close()

output = template('make\_table', rows=result)

return output

...

我们添加了两样东西。首先我们从Bottle中导入了 template 函数以使用模板功能，接着，我们渲染 make\_table 这个模板(参数是rows=result)，把模板函数的返回值赋予 output 变量，并返回 output 。如有必要，我们可添加更多的参数。

返回一个字符串的列表模板，这样就无需任何转换。 只需一行代码： return template('make\_table', rows=result)，就能实现上述的结果。

对应的模板文件。

%#template to generate a HTML table from a list of tuples (or list of lists, or tuple of tuples or ...)

<**p**>The open items are as follows:</**p**>

<**table** border="1">

%for row in rows:

<**tr**>

%for col in row:

<**td**>{{col}}</**td**>

%end

</**tr**>

%end

</**table**>

将上面的代码保存为 make\_table.tpl 文件，和 todo.py 放在同一个目录。

让我们看看这个代码：每一行以 % 开头被解释为 Python 代码，因为它是有效的 Python，也只允许有效的Python语句。否则会 Python 异常，其它的则是普通的HTML标记。

如你所见，为了遍历 rows ，我们两次使用了Python的 for 语句。 rows 是持有查询结果的变量，一个元组的列表。第一个 for 语句遍历了列表中所有的元组，第二个 for 语句遍历了元组中的元素，将其放进表格中。 for , if , while 语句都需要通过 %end 来关闭，要不会得到不正确的结果。

如果想要在不以%开头的行中访问变量，则需要把它放在两个大括号中间。这告诉模板，需要用变量的实际值将其替换掉。

再次运行该脚本并查看输出。还没有完全好，但至少比那个元组的列表更具可读的。你可以添加上述非常简单的HTML标记，还可以通过使用内联样式(css)来得到一个好看的输出。

**使用GET和POST**

能够查看所有代码事项后，让我们进入到下一步，添加新的待办事项到列表中。新的待办事项应该在一个常规的HTML表单中，通过GET方式提交。

让我们先来添加一个接受GET请求的route。

from bottle import route, run, debug, template, request

...

return template('make\_table', rows=result)

...

@route('/new', method='GET')

def new\_item():

new = request.GET.get('task', '').strip()

conn = sqlite3.connect('todo.db')

c = conn.cursor()

c.execute("INSERT INTO todo (task,status) VALUES (?,?)", (new,1))

new\_id = c.lastrowid

conn.commit()

c.close()

return '<**p**>The new task was inserted into the database, the ID is %s</**p**>' % new\_id

为了访问GET(或POST)中的数据，我们需要从Bottle中导入 request ，通过 request.GET.get('task', '').strip() 来获取表单中 task 字段的数据。可多次使用 request.GET.get() 来获取表单中所有字段的数据。

接下来是对数据的操作：写入数据库，获取返回的ID，生成页面。

因为我们是从HTML表单中获取数据，所以现在让我们来创建这个表单吧。我们通过 /new 这个URL来添加待办事项。

代码需要扩展如下:

...

@route('/new', method='GET')

def new\_item():

if request.GET.get('save','').strip():

new = request.GET.get('task', '').strip()

conn = sqlite3.connect('todo.db')

c = conn.cursor()

c.execute("INSERT INTO todo (task,status) VALUES (?,?)", (new,1))

new\_id = c.lastrowid

conn.commit()

c.close()

return '<**p**>The new task was inserted into the database, the ID is %s</**p**>' % new\_id

else:

return template('new\_task.tpl')

对应的 new\_task.tpl 模板如下。

<**p**>Add a new task to the ToDo list:</**p**>

<**form** action="/new" method="GET">

<**input** type="text" size="100" maxlength="100" name="task">

<**input** type="submit" name="save" value="save">

</**form**>

如你所见，这个模板只是纯HTML的，不包含Python代码。

这样，我们就完成了添加待办事项这个功能。

如果你想通过POST来获取数据，那么用 request.POST.get() 来代替 request.GET.get() 就行了。

**修改已有待办事项**

最后，我们需要做的是修改已有待办事项。

仅使用我们当前了解到的route类型，是可以完成这个任务的，但太取巧了。Bottle还提供了一种 动态route ，可以更简单地实现。

基本的动态route声明如下:

@route('/myroute/<**something**>')

关键的区别在于那个冒号。它告诉了Bottle，在下一个 / 之前， :something 可以匹配任何字符串。 :something 匹配到的字符串会传递给回调函数，进一步地处理。

我们的ToDo列表，我们将创建一个路由 @route('/edit/<no:int>), 这里的 no 是ID(integer)

对应的代码如下。

@route('/edit/<**no:int**>', method='GET')

def edit\_item(no):

if request.GET.get('save','').strip():

edit = request.GET.get('task','').strip()

status = request.GET.get('status','').strip()

if status == 'open':

status = 1

else:

status = 0

conn = sqlite3.connect('todo.db')

c = conn.cursor()

c.execute("UPDATE todo SET task = ?, status = ? WHERE id LIKE ?", (edit, status, no))

conn.commit()

return '<**p**>The item number %s was successfully updated</**p**>' % no

else:

conn = sqlite3.connect('todo.db')

c = conn.cursor()

c.execute("SELECT task FROM todo WHERE id LIKE ?", (str(no)))

cur\_data = c.fetchone()

return template('edit\_task', old=cur\_data, no=no)

以上基本就是我们添加新项目要做的了，比如使用 GET 数据。这里使用的动态路由 <no:int>，将数字传递给相应的函数，如你所见， no 是整数ID在函数内使用，以访问正确的数据库内的数据的行。

对应的 edit\_task.tpl 模板如下。

%#template for editing a task

%#the template expects to receive a value for "no" as well a "old", the text of the selected ToDo item

<**p**>Edit the task with ID = {{no}}</**p**>

<**form** action="/edit/{{no}}" method="get">

<**input** type="text" name="task" value="{{old[0]}}" size="100" maxlength="100">

<**select** name="status">

<**option**>open</**option**>

<**option**>closed</**option**>

</**select**>

<**br**/>

<**input** type="submit" name="save" value="save">

</**form**>

再一次，模板中混合了HTML代码和Python代码，之前已解释过。

你也可在动态route中使用正则表达式，稍后会提及。

**在动态route中使用正则表达式**

Bottle允许在动态route中使用正则表达式。

我们假设需要通过 item1 这样的形式来访问数据库中id为1的待办事项。显然，我们不想为每个待办事项都创建一个route。鉴于route中的”item”部分是固定的，简单的route就无法满足需求了，我们需要在route中使用正则表达式。

使用正则表达式的解决方法如下。

@route('/item<item:re:[0-9]+>')

def show\_item(item):

conn = sqlite3.connect('todo.db')

c = conn.cursor()

c.execute("SELECT task FROM todo WHERE id LIKE ?", (item))

result = c.fetchall()

c.close()

if not result:

return 'This item number does not exist!'

else:

return 'Task: %s' %result[0]

这个例子是人工构造的--根据实验这将是一个简单的动态路由，不过，我们想看看正则表达式工作方式：这行 @route(/item<item\_:re:[0-9]+>) 是一个普通的路由,但 # 包围部分解释为正则表达式，这是动态路由的一部分。这里，我们要匹配0-9之间的数字， 函数 “show\_item” 就在数据库中检查是否存在给定的项目。如果它存在返回相应的内容。如你所见，路由的正则表达式部分被转发。此外，像在这种情况下， 它总是被转发为一个字符串，即使它是一个普通的整数。

**返回静态文件**

有时候，我们只是想返回已有的静态文件。例如我们的应用中有个静态的帮助页面help.html，我们不希望每次访问帮助页面的时候都动态生成。

from bottle import route, run, debug, template, request, static\_file

@route('/help')

def help():

return static\_file('help.html', root='/path/to/file')

首先，我们需要从Bottle中导入 static\_file 函数。它接受至少两个参数，一个是需要返回的文件的文件名，一个是该文件的路径。即使该文件和你的应用在同一个目录下，还是要指定文件路径(可以使用”.”)。Bottle会猜测文件的MIME类型，并自动设置。如果你想显式指定MIME类型，可以在static\_file函数里面加上例如 mimetype='text/html' 这样的参数。 static\_file 函数可和任何route配合使用，包括动态route。

**返回JSON数据**

有时我们希望返回JSON，以便在客户端使用JavaScript来生成页面，Bottle直接支持返回JSON数据。JSON似乎已经是Web应用之间交换数据的标准格式了。更进一步，JSON可以被很多语言解析处理，包括Python。

我们假设现在需要返回JSON数据。

@route('/json<json:re:[0-9]+>')

def show\_json(json):

conn = sqlite3.connect('todo.db')

c = conn.cursor()

c.execute("SELECT task FROM todo WHERE id LIKE ?", (json))

result = c.fetchall()

c.close()

if not result:

return {'task':'This item number does not exist!'}

else:

return {'Task': result[0]}

很简单，只需要返回一个Python中的字典就可以了，Bottle会自动将其转换为JSON，再传输到客户端。如果你访问”<http://localhost/json1>“，你应能得到 {"Task": ["Read A-byte-of-python to get a good introduction into Python"]} 类型的JSON数据。

**捕获错误**

为了避免用户看到出错信息，我们需要捕获应用运行时出现的错误，以提供更友好的错误提示。Bottle提供了专门用于捕获错误的route。

例如，我们想捕获403错误。

from bottle import error

@error(403)

def mistake(code):

return 'The parameter you passed has the wrong format!'

首先，我们需要从Bottle中导入 error ，然后通过 error(403) 来定义创建一个route，用于捕获所有”403 forbidden”错误。注意，该route总是会将error-code传给 mistake() 函数，即使你不需要它。所以回调函数至少要接受一个参数，否则会失效。

一样的，同一个回调函数可以捕获多种错误。

@error(404)

@error(403)

def mistake(code):

return 'There is something wrong!'

效果和下面一样。

@error(403)

def mistake403(code):

return 'The parameter you passed has the wrong format!'

@error(404)

def mistake404(code):

return 'Sorry, this page does not exist!'

**总结**

通过以上章节，你应该对Bottle框架有了一个大致的了解，可以使用Bottle进行开发了。

接下来的章节会简单介绍一下，如何在大型项目中使用Bottle。此外，我们还会介绍如何将Bottle部署到更高性能的Web服务器上。

[安装服务器](http://bottle.zzir.cn/index.html#id5)[¶](http://bottle.zzir.cn/index.html#server-setup)

到目前为止，我们还是使用Bottle内置的，随Python一起发布的 [WSGI reference Server](http://docs.python.org/library/wsgiref.html#module-wsgiref.simple_server) 服务器。尽管该服务器十分适合用于开发环境，但是它确实不适用于大项目。在我们介绍其他服务器之前，我们先看看如何优化内置服务器的设置。

**更改服务器的端口和IP**

默认的，Bottle会监听127.0.0.1(即 localhost )的 8080 端口。

如果要更改该设置，更改 run 函数的参数即可。

run(port=80)

更改端口，监听80端口

更改监听的IP地址

run(host='123.45.67.89')

If needed, both parameters can be combined, like:

run(port=80, host='123.45.67.89')

当Bottle运行在其他服务器上面时， port 和 host 参数依然适用，稍后会介绍。

**在其他服务器上运行**

在大型项目上，Bottle自带的服务器会成为一个性能瓶颈，因为它是单线程的，一次只能响应一个请求。

Bottle已经可以工作在很多多线程的服务器上面了，例如 [Cherrypy](http://www.cherrypy.org/), [Fapws3](http://github.com/william-os4y/fapws3), [Flup](http://trac.saddi.com/flup) 和 [Paste](http://pythonpaste.org/) ，所以我们建议在大型项目上使用高性能的服务器。

如果想运行在Paste服务器上面，代码如下(译者注：需要先安装Paste)。

from bottle import PasteServer

...

run(server=PasteServer)

其他服务器如 FlupServer, CherryPyServer 和 FapwsServer 也类似。

**使用** [**mod\_wsgi**](http://code.google.com/p/modwsgi/) **运行在Apache上**

或许你已经有了一个 [Apache](http://www.apache.org/) 服务器，那么可以考虑使用 [mod\_wsgi](http://code.google.com/p/modwsgi/) 。

我们假设你的Apache已经能跑起来，且mod\_wsgi也能工作了。在很多Linux发行版上，都能通过包管理软件简单地安装mod\_wsgi。

Bottle已经自带用于mod\_wsgi的适配器，所以让Bottle跑在mod\_wsgi上面是很简单的。

接下来的例子里，我们假设你希望通过 http://www.mypage.com/todo 来访问”ToDo list”这个应用，且代码、模板、和SQLite数据库存放在 /var/www/todo 目录。

如果通过mod\_wsgi来运行你应用，那么必须从代码中移除 run() 函数。

然后，创建一个 adapter.wsgi 文件，内容如下。

import sys, os, bottle

sys.path = ['/var/www/todo/'] + sys.path

os.chdir(os.path.dirname(\_\_file\_\_))

import todo # This loads your application

application = bottle.default\_app()

将其保存到 /var/www/todo 目录下面。其实，可以给该文件起任何名字，只要后缀名为 .wsgi 即可。

最后，我们需要在Apache的配置中添加一个虚拟主机。

<VirtualHost \*>

ServerName mypage.com

WSGIDaemonProcess todo user=www-data group=www-data processes=1 threads=5

WSGIScriptAlias / /var/www/todo/adapter.wsgi

<Directory /var/www/todo>

WSGIProcessGroup todo

WSGIApplicationGroup %{GLOBAL}

Order deny,allow

Allow from all

</Directory>

</VirtualHost>

重启Apache服务器后，即可通过 http://www.mypage.com/todo 来访问你的应用。

[结语](http://bottle.zzir.cn/index.html#id6)[¶](http://bottle.zzir.cn/index.html#final-words)

现在，我们这个教程已经结束了。我们学习了Bottle的基础知识，然后使用Bottle来写了第一个应用。另外，我们还介绍了如何在大型项目中使用Bottle，以及使用mod\_wsgi在Apache中运行Bottle应用。

我们并没有在这份教程里介绍Bottle的方方面面。我们没有介绍如何上传文件，验证数据的可靠性。还有，我们也没介绍如何在模板中调用另一个模板。以上，可以在 [Bottle documentation](http://bottlepy.org/docs/dev/tutorial.html) 中找到答案。

[完整代码](http://bottle.zzir.cn/index.html#id7)[¶](http://bottle.zzir.cn/index.html#complete-example-listing)

我们是一步一步地开发待办事项列表的，这里是完整的代码。

todo.py

import sqlite3

from bottle import route, run, debug, template, request, static\_file, error

# only needed when you run Bottle on mod\_wsgi

from bottle import default\_app

@route('/todo')

def todo\_list():

conn = sqlite3.connect('todo.db')

c = conn.cursor()

c.execute("SELECT id, task FROM todo WHERE status LIKE '1';")

result = c.fetchall()

c.close()

output = template('make\_table', rows=result)

return output

@route('/new', method='GET')

def new\_item():

if request.GET.get('save','').strip():

new = request.GET.get('task', '').strip()

conn = sqlite3.connect('todo.db')

c = conn.cursor()

c.execute("INSERT INTO todo (task,status) VALUES (?,?)", (new,1))

new\_id = c.lastrowid

conn.commit()

c.close()

return '<p>The new task was inserted into the database, the ID is %s</p>' % new\_id

else:

return template('new\_task.tpl')

@route('/edit/<no:int>', method='GET')

def edit\_item(no):

if request.GET.get('save','').strip():

edit = request.GET.get('task','').strip()

status = request.GET.get('status','').strip()

if status == 'open':

status = 1

else:

status = 0

conn = sqlite3.connect('todo.db')

c = conn.cursor()

c.execute("UPDATE todo SET task = ?, status = ? WHERE id LIKE ?", (edit,status,no))

conn.commit()

return '<p>The item number %s was successfully updated</p>' %no

else:

conn = sqlite3.connect('todo.db')

c = conn.cursor()

c.execute("SELECT task FROM todo WHERE id LIKE ?", (str(no)))

cur\_data = c.fetchone()

return template('edit\_task', old = cur\_data, no = no)

@route('/item<item:re:[0-9]+>')

def show\_item(item):

conn = sqlite3.connect('todo.db')

c = conn.cursor()

c.execute("SELECT task FROM todo WHERE id LIKE ?", (item))

result = c.fetchall()

c.close()

if not result:

return 'This item number does not exist!'

else:

return 'Task: %s' %result[0]

@route('/help')

def help():

static\_file('help.html', root='.')

@route('/json<json:re:[0-9]+>')

def show\_json(json):

conn = sqlite3.connect('todo.db')

c = conn.cursor()

c.execute("SELECT task FROM todo WHERE id LIKE ?", (json))

result = c.fetchall()

c.close()

if not result:

return {'task':'This item number does not exist!'}

else:

return {'Task': result[0]}

@error(403)

def mistake403(code):

return 'There is a mistake in your url!'

@error(404)

def mistake404(code):

return 'Sorry, this page does not exist!'

debug(True)

run(reloader=True)

#remember to remove reloader=True and debug(True) when you move your application from development to a productive environment

make\_table.tpl模板

%#template to generate a HTML table from a list of tuples (or list of lists, or tuple of tuples or ...)

<**p**>The open items are as follows:</**p**>

<**table** border="1">

%for row in rows:

<**tr**>

%for col in row:

<**td**>{{col}}</**td**>

%end

</**tr**>

%end

</**table**>

Template edit\_task.tpl:

%#template for editing a task

%#the template expects to receive a value for "no" as well a "old", the text of the selected ToDo item

<**p**>Edit the task with ID = {{no}}</**p**>

<**form** action="/edit/{{no}}" method="get">

<**input** type="text" name="task" value="{{old[0]}}" size="100" maxlength="100">

<**select** name="status">

<**option**>open</**option**>

<**option**>closed</**option**>

</**select**>

<**br**/>

<**input** type="submit" name="save" value="save">

</**form**>

new\_task.tpl 模板

%#template for the form for a new task

<**p**>Add a new task to the ToDo list:</**p**>

<**form** action="/new" method="GET">

<**input** type="text" size="100" maxlength="100" name="task">

<**input** type="submit" name="save" value="save">

</**form**>

异步应用入门[¶](http://bottle.zzir.cn/index.html#primer-to-asynchronous-applications)

异步设计模式和 [WSGI](http://www.python.org/dev/peps/pep-3333/) 的同步本质并不能很好地兼容。这就是为什么大部分的异步框架(tornado, twisted, ...)都实现了专有的API来暴露它们的异步特征。Bottle是一个WSGI框架，也继承了WSGI的同步本质，但是谢谢优秀的 [gevent项目](http://www.gevent.org/) ，我们可以使用Bottle来编写异步应用。这份文档介绍了在Bottle中如何使用异步WSGI。

同步WSGI的限制[¶](http://bottle.zzir.cn/index.html#the-limits-of-synchronous-wsgi)

简单来说， [WSGI标准 (pep 3333)](http://www.python.org/dev/peps/pep-3333/) 定义了下面这一个request/response的循环：每次请求到达的时候，应用中的callable会被调用一次，返回一个主体iterator。接着服务器会遍历该主体，分块写入socket。遍历完整个主体，就关闭客户端的连接。

足够简单，但是存在一个小问题：所有这些都是同步的。如果你的应用需要等待数据(IO, socket, 数据库, ...)，除了返回一个空字符串(忙等)，就只能阻塞当前线程。两种办法都会占用当前线程，导致线程不能处理新的请求，只能处理当前的一个请求。

大部分服务器都限制了线程的数量，避免伴随它们而来的资源消耗。常见的是一个线程池，内有20个或更少数量的线程。一旦所有的线程都被占用了，任何新的请求都会阻塞。事实上，对于其他人来说，服务器已经宕机了。如果你想实现一个聊天程序，使用ajax轮询来获取实时消息，很快你就会受到线程数量的限制。这样能同时服务的用户就太少了。

救星，Greenlet[¶](http://bottle.zzir.cn/index.html#greenlets-to-the-rescue)

大多数服务器的线程池都限制了线程池中线程的数量，避免创建和切换线程的代价。尽管和创建进程(fork)的代价比起来，线程还是挺便宜的。但是也没便宜到可以接受为每一个请求创建一个线程。

[gevent](http://www.gevent.org/) 模块添加了 *greenlet* 的支持。greenlet和传统的线程类似，但其创建只需消耗很少的资源。基于gevent的服务器可以生成成千上万的greenlet，为每个连接分配一个greenlet也毫无压力。阻塞greenlet，也不会影响到服务器接受新的请求。同时处理的连接数理论上是没有限制的。

这令创建异步应用难以置信的简单，因为它们看起来很想同步程序。基于gevent服务器实际上不是异步的，是大规模多线程。下面是一个例子。

from gevent import monkey; monkey.patch\_all()

from time import sleep

from bottle import route, run

@route('/stream')

def stream():

yield 'START'

sleep(3)

yield 'MIDDLE'

sleep(5)

yield 'END'

run(host='0.0.0.0', port=8080, server='gevent')

第一行很重要。它让gevent monkey-patch了大部分Python的阻塞API，让它们不阻塞当前线程，将CPU让给下一个greenlet。它实际上用基于gevent的伪线程替换了Python的线程。这就是你依然可以使用 time.sleep() 这个照常来说会阻塞线程的函数。如果这种monkey-patch的方式感令你感到不舒服，你依然可以使用gevent中相应的函数 gevent.sleep() 。

如果你运行了上面的代码，接着访问 http://localhost:8080/stream ，你可看到 *START*, *MIDDLE*, 和 *END* 这几个字样依次出现(用时大约8秒)。它像普通的线程一样工作，但是现在你的服务器能同时处理成千上万的连接了。

注解

一些浏览器在开始渲染一个页面之前，会缓存确定容量的数据。在这些浏览器上，你需要返回更多的数据才能看到效果。另外，很多浏览器限制一个URL只使用一个连接。在这种情况下，你可以使用另外的浏览器，或性能测试工具(例如： *ab* 或 *httperf* )来测试性能。

事件回调函数[¶](http://bottle.zzir.cn/index.html#event-callbacks)

异步框架的常见设计模式(包括 tornado, twisted, node.js 和 friends)，是使用非阻塞的API，绑定回调函数到异步事件上面。在显式地关闭之前，socket会保持打开，以便稍后回调函数往socket里面写东西。下面是一个基于 [tornado](http://www.tornadoweb.org/documentation#non-blocking-asynchronous-requests) 的例子。

class MainHandler(tornado.web.RequestHandler):

@tornado.web.asynchronous

def get(self):

worker = SomeAsyncWorker()

worker.on\_data(lambda chunk: self.write(chunk))

worker.on\_finish(lambda: self.finish())

主要的好处就是MainHandler能早早结束，在回调函数继续写socket来响应之前的请求的时候，当前线程能继续接受新的请求。这样就是为什么这类框架能同时处理很多请求，只使用很少的操作系统线程。

对于Gevent和WSGI来说，情况就不一样了：首先，早早结束没有好处，因为我们的(伪)线程池已经没有限制了。第二，我们不能早早结束，因为这样会关闭socket(WSGI要求如此)。第三，我们必须返回一个iterator，以遵守WSGI的约定。

为了遵循WSGI规范，我们只需返回一个iterable的实体，异步地将其写回客户端。在 [gevent.queue](http://www.gevent.org/gevent.queue.html) 的帮助下，我们可以 *模拟* 一个脱管的socket，上面的例子可写成这样。

@route('/fetch')

def fetch():

body = gevent.queue.Queue()

worker = SomeAsyncWorker()

worker.on\_data(body.put)

worker.on\_finish(lambda: body.put(StopIteration))

worker.start()

return body

从服务器的角度来看，queue对象是iterable的。如果为空，则阻塞，一旦遇到 StopIteration 则停止。这符合WSGI规范。从应用的角度来看，queue对象表现的像一个不会阻塞socket。你可以在任何时刻写入数据，pass it around，甚至启动一个新的(伪)线程，异步写入。这是在大部分情况下，实现长轮询。

最后： WebSockets[¶](http://bottle.zzir.cn/index.html#finally-websockets)

让我们暂时忘记底层的细节，来谈谈WebSocket。既然你正在阅读这篇文章，你有可能已经知道什么是WebSocket了，一个在浏览器(客户端)和Web应用(服务端)的双向的交流通道。

感谢 [gevent-websocket](http://pypi.python.org/pypi/gevent-websocket/) 包帮我们做的工作。下面是一个WebSocket的简单例子，接受消息然后将其发回客户端。

from bottle import request, Bottle, abort

app = Bottle()

@app.route('/websocket')

def handle\_websocket():

wsock = request.environ.get('wsgi.websocket')

if not wsock:

abort(400, 'Expected WebSocket request.')

while True:

try:

message = wsock.receive()

wsock.send("Your message was: %r" % message)

except WebSocketError:

break

from gevent.pywsgi import WSGIServer

from geventwebsocket import WebSocketError

from geventwebsocket.handler import WebSocketHandler

server = WSGIServer(("0.0.0.0", 8080), app,

handler\_class=WebSocketHandler)

server.serve\_forever()

while循环直到客户端关闭连接的时候才会终止。You get the idea :)

客户端的JavaScript API也十分简洁明了:

<!DOCTYPE html>

<**html**>

<**head**>

<**script** type="text/javascript">

**var** ws = **new** WebSocket("ws://example.com:8080/websocket");

ws.onopen = **function**() {

ws.send("Hello, world");

};

ws.onmessage = **function** (evt) {

alert(evt.data);

};

</**script**>

</**head**>

</**html**>

秘诀[¶](http://bottle.zzir.cn/index.html#recipes)

这里收集了一些常见用例的代码片段和例子.

使用Session[¶](http://bottle.zzir.cn/index.html#keeping-track-of-sessions)

Bottle自身并没有提供Session的支持，因为在一个迷你框架里面，没有合适的方法来实现。根据需求和使用环境，你可以使用 [beaker](http://beaker.groovie.org/) 中间件或自己来实现。下面是一个使用beaker的例子，Session数据存放在”./data”目录里面:

import bottle

from beaker.middleware import SessionMiddleware

session\_opts = {

'session.type': 'file',

'session.cookie\_expires': 300,

'session.data\_dir': './data',

'session.auto': True

}

app = SessionMiddleware(bottle.app(), session\_opts)

@bottle.route('/test')

def test():

s = bottle.request.environ.get('beaker.session')

s['test'] = s.get('test',0) + 1

s.save()

return 'Test counter: %d' % s['test']

bottle.run(app=app)

Debugging with Style: 调试中间件[¶](http://bottle.zzir.cn/index.html#debugging-with-style-debugging-middleware)

Bottle捕获所有应用抛出的异常，防止异常导致WSGI服务器崩溃。如果内置的 [**debug()**](http://bottle.zzir.cn/index.html#bottle.debug) 模式不能满足你的要求，你想在你自己写的中间件里面处理这些异常，那么你可以关闭这个功能。

import bottle

app = bottle.app()

app.catchall = False #Now most exceptions are re-raised within bottle.

myapp = DebuggingMiddleware(app) #Replace this with a middleware of your choice (see below)

bottle.run(app=myapp)

现在，Bottle仅会捕获并处理它自己抛出的异常( [**HTTPError**](http://bottle.zzir.cn/index.html#bottle.HTTPError) , [**HTTPResponse**](http://bottle.zzir.cn/index.html#bottle.HTTPResponse) 和 [**BottleException**](http://bottle.zzir.cn/index.html#bottle.BottleException) )，你的中间件可以处理剩下的那些异常。

[werkzeug](http://werkzeug.pocoo.org/documentation/dev/debug.html) 和 [paste](http://pythonpaste.org/modules/evalexception.html) 这两个第三方库都提供了非常强大的调试中间件。如果是 [werkzeug](http://werkzeug.pocoo.org/documentation/dev/debug.html) ，可看看 [**werkzeug.debug.DebuggedApplication**](http://werkzeug.pocoo.org/docs/debug/#werkzeug.debug.DebuggedApplication) ，如果是 [paste](http://pythonpaste.org/modules/evalexception.html) ，可看看 **paste.evalexception.middleware.EvalException** 。它们都可让你检查运行栈，甚至在保持运行栈上下文的情况下，执行Python代码。所以 **不要在生产环境中使用它们** 。

Bottle 应用单元测试[¶](http://bottle.zzir.cn/index.html#unit-testing-bottle-applications)

Unit测试一般用于测试应用中的函数，但不需要一个WSGI环境。

使用 [Nose](http://readthedocs.org/docs/nose) 的简单例子。

import bottle

@bottle.route('/')

def index():

return 'Hi!'

if \_\_name\_\_ == '\_\_main\_\_':

bottle.run()

测试代码:

import mywebapp

def test\_webapp\_index():

assert mywebapp.index() == 'Hi!'

在这个例子中，Bottle的route()函数没有被执行，仅测试了index()函数。

功能测试[¶](http://bottle.zzir.cn/index.html#functional-testing-bottle-applications)

任何基于HTTP的测试系统都可用于测试WSGI服务器，但是有些测试框架与WSGI服务器工作得更好。它们可以在一个可控环境里运行WSGI应用，充分利用traceback和调试工具。 [Testing tools for WSGI](http://www.wsgi.org/en/latest/testing.html) 是一个很好的上手工具。

使用 [WebTest](http://webtest.pythonpaste.org/) 和 [Nose](http://readthedocs.org/docs/nose) 的例子。

from webtest import TestApp

import mywebapp

def test\_functional\_login\_logout():

app = TestApp(mywebapp.app)

app.post('/login', {'user': 'foo', 'pass': 'bar'}) # log in and get a cookie

assert app.get('/admin').status == '200 OK' # fetch a page successfully

app.get('/logout') # log out

app.reset() # drop the cookie

# fetch the same page, unsuccessfully

assert app.get('/admin').status == '401 Unauthorized'

嵌入其他WSGI应用[¶](http://bottle.zzir.cn/index.html#embedding-other-wsgi-apps)

并不建议你使用这个方法，你应该在Bottle前面使用一个中间件来做这样的事情。但你确实可以在Bottle里面调用其他WSGI应用，让Bottle扮演一个中间件的角色。下面是一个例子。

from bottle import request, response, route

subproject = SomeWSGIApplication()

@route('/subproject/:subpath#.\*#', method='ANY')

def call\_wsgi(subpath):

new\_environ = request.environ.copy()

new\_environ['SCRIPT\_NAME'] = new\_environ.get('SCRIPT\_NAME','') + '/subproject'

new\_environ['PATH\_INFO'] = '/' + subpath

def start\_response(status, headerlist):

response.status = int(status.split()[0])

for key, value in headerlist:

response.add\_header(key, value)

return app(new\_environ, start\_response)

再次强调，并不建议使用这种方法。之所以介绍这种方法，是因为很多人问起，如何在Bottle中调用WSGI应用。

忽略尾部的反斜杠[¶](http://bottle.zzir.cn/index.html#ignore-trailing-slashes)

在Bottle看来， /example 和 /example/ 是两个不同的route [[1]](http://bottle.zzir.cn/index.html#id3) 。为了一致对待这两个URL，你应该添加两个route。

@route('/test')

@route('/test/')

def test(): return 'Slash? no?'

给所有 URL 以 '/' 结尾添加WSGI中间件:

class StripPathMiddleware(object):

def \_\_init\_\_(self, app):

self.app = app

def \_\_call\_\_(self, e, h):

e['PATH\_INFO'] = e['PATH\_INFO'].rstrip('/')

return self.app(e,h)

app = bottle.app()

myapp = StripPathMiddleware(app)

bottle.run(app=myapp)

或添加 before\_request 钩子：

@hook('before\_request')

def strip\_path():

request.environ['PATH\_INFO'] = request.environ['PATH\_INFO'].rstrip('/')

**脚注**

|  |  |
| --- | --- |
| [[1]](http://bottle.zzir.cn/index.html#id2) | 因为确实如此，见 <<http://www.ietf.org/rfc/rfc3986.txt>> |

Requests保持连接[¶](http://bottle.zzir.cn/index.html#keep-alive-requests)

注解

详见 [异步应用入门](http://bottle.zzir.cn/index.html#document-async) 。

像XHR这样的”push”机制，需要在HTTP响应头中加入 “Connection: keep-alive” ，以便在不关闭连接的情况下，写入响应数据。WSGI并不支持这种行为，但如果在Bottle中使用 [gevent](http://www.gevent.org/) 这个异步框架，还是可以实现的。下面是一个例子，可配合 [gevent](http://www.gevent.org/) HTTP服务器或 [paste](http://pythonpaste.org/modules/evalexception.html) HTTP服务器使用(也许支持其他服务器，但是我没试过)。在run()函数里面使用 server='gevent' 或 server='paste' 即可使用这两种服务器。

from gevent import monkey; monkey.patch\_all()

import gevent

from bottle import route, run

@route('/stream')

def stream():

yield 'START'

gevent.sleep(3)

yield 'MIDDLE'

gevent.sleep(5)

yield 'END'

run(host='0.0.0.0', port=8080, server='gevent')

通过浏览器访问 http://localhost:8080/stream ，可看到’START’，’MIDDLE’，和’END’这三个字眼依次出现，一共用了8秒。

Gzip压缩[¶](http://bottle.zzir.cn/index.html#gzip-compression-in-bottle)

注解

详见 [compression](https://github.com/bottlepy/bottle/issues/92)

Gzip压缩，可加速网站静态资源(例如CSS和JS文件)的访问。人们希望Bottle支持Gzip压缩，（但是不支持)......

支持Gzip压缩并不简单，一个合适的Gzip实现应该满足以下条件。

* 压缩速度要快
* 如果浏览器不支持，则不压缩
* 不压缩那些已经充分压缩的文件(图像，视频)
* 不压缩动态文件
* 支持两种压缩算法(gzip和deflate)
* 缓存那些不经常变化的压缩文件
* 不验证缓存中那些已经变化的文件(De-validate the cache if one of the files changed anyway)
* 确保缓存不太大
* 不缓存小文件，因为寻道时间或许比压缩时间还长

因为有上述种种限制，建议由WSGI服务器来处理Gzip压缩而不是Bottle。像 [cherrypy](http://www.cherrypy.org/) 就提供了一个 [GzipFilter](http://www.cherrypy.org/wiki/GzipFilter) 中间件来处理Gzip压缩。

使用钩子[¶](http://bottle.zzir.cn/index.html#using-the-hooks-plugin)

例如，你想提供跨域资源共享，可参考下面的例子。

from bottle import hook, response, route

@hook('after\_request')

def enable\_cors():

response.headers['Access-Control-Allow-Origin'] = '\*'

@route('/foo')

def say\_foo():

return 'foo!'

@route('/bar')

def say\_bar():

return {'type': 'friendly', 'content': 'Hi!'}

你也可以使用 before\_request ，这样在route的回调函数被调用之前，都会调用你的钩子函数。

在Heroku中使用Bottle[¶](http://bottle.zzir.cn/index.html#using-bottle-with-heroku)

[Heroku](http://heroku.com/) ，一个流行的云应用平台，提供Python支持。

这份教程基于 [Heroku Quickstart](http://devcenter.heroku.com/articles/quickstart), 用Bottle特有的代码替换了 [Getting Started with Python on Heroku/Cedar](http://devcenter.heroku.com/articles/python) 中的 Write Your App 在 <<http://devcenter.heroku.com/articles/python#write_your_app>>这部分:

import os

from bottle import route, run

@route("/")

def hello\_world():

return "Hello World!"

run(host="0.0.0.0", port=int(os.environ.get("PORT", 5000)))

Heroku使用 *os.environ* 字典来提供Bottle应用需要监听的端口。

常见问题[¶](http://bottle.zzir.cn/index.html#frequently-asked-questions)

关于Bottle[¶](http://bottle.zzir.cn/index.html#about-bottle)

Bottle适合用于复杂的应用吗？[¶](http://bottle.zzir.cn/index.html#is-bottle-suitable-for-complex-applications)

Bottle是一个 *迷你* 框架，被设计来提供原型开发和创建小的Web应用和服务。它能快速上手，并帮助你快速完成任务，但缺少一些高级功能和一些其他框架已提供的已知问题解决方法(例如:MVC， ORM，表单验证，手脚架，XML-RPC)。尽管 *可以* 添加这些功能，然后通过Bottle来开发复杂的应用，但我们还是建议你使用一些功能完备的Web框架，例如 [pylons](http://pylonshq.com/) 或 [paste](http://pythonpaste.org/modules/evalexception.html) 。

常见的，意料之外的问题[¶](http://bottle.zzir.cn/index.html#common-problems-and-pitfalls)

在mod\_wsgi/mod\_python中的 “Template Not Found” 错误[¶](http://bottle.zzir.cn/index.html#template-not-found-in-mod-wsgi-mod-python)

Bottle会在 ./ 和 ./views/ 目录中搜索模板。在一个 [mod\_python](http://www.modpython.org/) 或 [mod\_wsgi](http://code.google.com/p/modwsgi/) 环境，当前工作目录( ./ )是由Apache的设置决定的。你应该在模板的搜索路径中添加一个绝对路径。

bottle.TEMPLATE\_PATH.insert(0,'/absolut/path/to/templates/')

这样，Bottle就能在正确的目录下搜索模板了

动态route和反斜杠[¶](http://bottle.zzir.cn/index.html#dynamic-routes-and-slashes)

在 [dynamic route syntax](http://bottle.zzir.cn/index.html#tutorial-dynamic-routes) 中， :name 匹配任何字符，直到出现一个反斜杠。工作方式相当与 [^/]+ 这样一个正则表达式。为了将反斜杠包涵进来，你必须在 :name 中添加一个自定义的正则表达式。例如： /images/:filepath#.\*# 会匹配 /images/icons/error.png ，但不匹配 /images/:filename 。

反向代理的问题[¶](http://bottle.zzir.cn/index.html#problems-with-reverse-proxies)

只用Bottle知道公共地址和应用位置的情况下，重定向和url-building才会起作用。(译者注：保留原文)Redirects and url-building only works if bottle knows the public address and location of your application.如果Bottle应用运行在反向代理或负载均衡后面，一些信息也许会丢失。例如， wsgi.url\_scheme 的值或 Host 头或许只能获取到代理的请求信息，而不是真实的用户请求。下面是一个简单的中间件代码片段，处理上面的问题，获取正确的值。

def fix\_environ\_middleware(app):

def fixed\_app(environ, start\_response):

environ['wsgi.url\_scheme'] = 'https'

environ['HTTP\_X\_FORWARDED\_HOST'] = 'example.com'

return app(environ, start\_response)

return fixed\_app

app = bottle.default\_app()

app.wsgi = fix\_environ\_middleware(app.wsgi)

开发和贡献[¶](http://bottle.zzir.cn/index.html#development-and-contribution)

这些章节是为那些对Bottle的开发和发布流程感兴趣的开发者准备的。

发布摘要和更改历史(不译)[¶](http://bottle.zzir.cn/index.html#release-notes-and-changelog)

Release 0.13[¶](http://bottle.zzir.cn/index.html#release-0-13)

* Added [**patch()**](http://bottle.zzir.cn/index.html#bottle.patch) shortcut for *route(..., method=’PATCH’)*

Release 0.12[¶](http://bottle.zzir.cn/index.html#release-0-12)

* New SimpleTemplate parser implementation
  + Support for multi-line code blocks (*<% ... %>*).
  + The keywords *include* and *rebase* are functions now and can accept variable template names.
* The new [**BaseRequest.route()**](http://bottle.zzir.cn/index.html#bottle.BaseRequest.route) property returns the [**Route**](http://bottle.zzir.cn/index.html#bottle.Route) that originally matched the request.
* Removed the BaseRequest.MAX\_PARAMS limit. The hash collision bug in CPythons dict() implementation was fixed over a year ago. If you are still using Python 2.5 in production, consider upgrading or at least make sure that you get security fixed from your distributor.
* New [**ConfigDict**](http://bottle.zzir.cn/index.html#bottle.ConfigDict) API (see [配置文件(初稿)](http://bottle.zzir.cn/index.html#document-configuration))

More information can be found in this [development blog post](http://blog.bottlepy.org/2013/07/19/preview-bottle-012.html).

Release 0.11[¶](http://bottle.zzir.cn/index.html#release-0-11)

* Native support for Python 2.x and 3.x syntax. No need to run 2to3 anymore.
* Support for partial downloads (Range header) in **static\_file()**.
* The new [**ResourceManager**](http://bottle.zzir.cn/index.html#bottle.ResourceManager) interface helps locating files bundled with an application.
* Added a server adapter for [waitress](http://docs.pylonsproject.org/projects/waitress/en/latest/).
* New [**Bottle.merge()**](http://bottle.zzir.cn/index.html#bottle.Bottle.merge) method to install all routes from one application into another.
* New [**BaseRequest.app**](http://bottle.zzir.cn/index.html#bottle.BaseRequest.app) property to get the application object that handles a request.
* Added [**FormsDict.decode()**](http://bottle.zzir.cn/index.html#bottle.FormsDict.decode) to get an all-unicode version (needed by WTForms).
* [**MultiDict**](http://bottle.zzir.cn/index.html#bottle.MultiDict) and subclasses are now pickle-able.

**API Changes**

* **Response.status** is a read-write property that can be assigned either a numeric status code or a status string with a reason phrase (200 OK). The return value is now a string to better match existing APIs (WebOb, werkzeug). To be absolutely clear, you can use the read-only properties [**BaseResponse.status\_code**](http://bottle.zzir.cn/index.html#bottle.BaseResponse.status_code) and [**BaseResponse.status\_line**](http://bottle.zzir.cn/index.html#bottle.BaseResponse.status_line).

**API Deprecations**

* **SimpleTALTemplate** is now deprecating. There seems to be no demand.

Release 0.10[¶](http://bottle.zzir.cn/index.html#release-0-10)

* Plugin API v2
  + To use the new API, set [**Plugin.api**](http://bottle.zzir.cn/index.html#bottle.Plugin.api) to 2.
  + [**Plugin.apply()**](http://bottle.zzir.cn/index.html#bottle.Plugin.apply) receives a [**Route**](http://bottle.zzir.cn/index.html#bottle.Route) object instead of a context dictionary as second parameter. The new object offers some additional information and may be extended in the future.
  + Plugin names are considered unique now. The topmost plugin with a given name on a given route is installed, all other plugins with the same name are silently ignored.
* The Request/Response Objects
  + Added [**BaseRequest.json**](http://bottle.zzir.cn/index.html#bottle.BaseRequest.json), [**BaseRequest.remote\_route**](http://bottle.zzir.cn/index.html#bottle.BaseRequest.remote_route), [**BaseRequest.remote\_addr**](http://bottle.zzir.cn/index.html#bottle.BaseRequest.remote_addr), [**BaseRequest.query**](http://bottle.zzir.cn/index.html#bottle.BaseRequest.query) and [**BaseRequest.script\_name**](http://bottle.zzir.cn/index.html#bottle.BaseRequest.script_name).
  + Added [**BaseResponse.status\_line**](http://bottle.zzir.cn/index.html#bottle.BaseResponse.status_line) and [**BaseResponse.status\_code**](http://bottle.zzir.cn/index.html#bottle.BaseResponse.status_code) attributes. In future releases, [**BaseResponse.status**](http://bottle.zzir.cn/index.html#bottle.BaseResponse.status) will return a string (e.g. 200 OK) instead of an integer to match the API of other common frameworks. To make the transition as smooth as possible, you should use the verbose attributes from now on.
  + Replaced [**MultiDict**](http://bottle.zzir.cn/index.html#bottle.MultiDict) with a specialized [**FormsDict**](http://bottle.zzir.cn/index.html#bottle.FormsDict) in many places. The new dict implementation allows attribute access and handles unicode form values transparently.
* 模板
  + Added three new functions to the SimpleTemplate default namespace that handle undefined variables: [**stpl.defined()**](http://bottle.zzir.cn/index.html#stpl.defined), [**stpl.get()**](http://bottle.zzir.cn/index.html#stpl.get) and [**stpl.setdefault()**](http://bottle.zzir.cn/index.html#stpl.setdefault).
  + The default escape function for SimpleTemplate now additionally escapes single and double quotes.
* Routing
  + A new route syntax (e.g. /object/<id:int>) and support for route wildcard filters.
  + Four new wildcard filters: *int*, *float*, *path* and *re*.
* Other changes
  + Added command line interface to load applications and start servers.
  + Introduced a [**ConfigDict**](http://bottle.zzir.cn/index.html#bottle.ConfigDict) that makes accessing configuration a lot easier (attribute access and auto-expanding namespaces).
  + Added support for raw WSGI applications to [**Bottle.mount()**](http://bottle.zzir.cn/index.html#bottle.Bottle.mount).
  + [**Bottle.mount()**](http://bottle.zzir.cn/index.html#bottle.Bottle.mount) parameter order changed.
  + [**Bottle.route()**](http://bottle.zzir.cn/index.html#bottle.Bottle.route) now accpets an import string for the callback parameter.
  + Dropped Gunicorn 0.8 support. Current supported version is 0.13.
  + Added custom options to Gunicorn server.
  + Finally dropped support for type filters. Replace with a custom plugin of needed.

Release 0.9[¶](http://bottle.zzir.cn/index.html#release-0-9)

**Whats new?**

* A brand new plugin-API. See [插件](http://bottle.zzir.cn/index.html#plugins) and [插件开发指南](http://bottle.zzir.cn/index.html#document-plugindev) for details.
* The [**route()**](http://bottle.zzir.cn/index.html#bottle.route) decorator got a lot of new features. See [**Bottle.route()**](http://bottle.zzir.cn/index.html#bottle.Bottle.route) for details.
* New server adapters for [gevent](http://www.gevent.org/), [meinheld](http://meinheld.org/) and [bjoern](https://github.com/jonashaag/bjoern).
* Support for SimpleTAL templates.
* Better runtime exception handling for mako templates in debug mode.
* Lots of documentation, fixes and small improvements.
* A new **Request.urlparts** property.

**Performance improvements**

* The **Router** now special-cases wsgi.run\_once environments to speed up CGI.
* Reduced module load time by ~30% and optimized template parser. See [8ccb2d](http://bottle.zzir.cn/commit/8ccb2d), [f72a7c](http://bottle.zzir.cn/commit/f72a7c) and [b14b9a](http://bottle.zzir.cn/commit/b14b9a) for details.
* Support for “App Caching” on Google App Engine. See [af93ec](http://bottle.zzir.cn/commit/af93ec).
* Some of the rarely used or deprecated features are now plugins that avoid overhead if the feature is not used.

**API changes**

This release is mostly backward compatible, but some APIs are marked deprecated now and will be removed for the next release. Most noteworthy:

* The static route parameter is deprecated. You can escape wild-cards with a backslash.
* Type-based output filters are deprecated. They can easily be replaced with plugins.

Release 0.8[¶](http://bottle.zzir.cn/index.html#release-0-8)

**API changes**

These changes may break compatibility with previous versions.

* The built-in Key/Value database is not available anymore. It is marked deprecated since 0.6.4
* The Route syntax and behaviour changed.
  + Regular expressions must be encapsulated with #. In 0.6 all non-alphanumeric characters not present in the regular expression were allowed.
  + Regular expressions not part of a route wildcard are escaped automatically. You don’t have to escape dots or other regular control characters anymore. In 0.6 the whole URL was interpreted as a regular expression. You can use anonymous wildcards (/index:#(\.html)?#) to achieve a similar behaviour.
* The BreakTheBottle exception is gone. Use [**HTTPResponse**](http://bottle.zzir.cn/index.html#bottle.HTTPResponse) instead.
* The [**SimpleTemplate**](http://bottle.zzir.cn/index.html#bottle.SimpleTemplate) engine escapes HTML special characters in {{bad\_html}} expressions automatically. Use the new {{!good\_html}} syntax to get old behaviour (no escaping).
* The [**SimpleTemplate**](http://bottle.zzir.cn/index.html#bottle.SimpleTemplate) engine returns unicode strings instead of lists of byte strings.
* bottle.optimize() and the automatic route optimization is obsolete.
* Some functions and attributes were renamed:
  + **Request.\_environ** is now **Request.environ**
  + **Response.header** is now **Response.headers**
  + [**default\_app()**](http://bottle.zzir.cn/index.html#bottle.default_app) is obsolete. Use [**app()**](http://bottle.zzir.cn/index.html#bottle.app) instead.
* The default **redirect()** code changed from 307 to 303.
* Removed support for @default. Use @error(404) instead.

**New features**

This is an incomplete list of new features and improved functionality.

* The [**Request**](http://bottle.zzir.cn/index.html#bottle.Request) object got new properties: **Request.body**, **Request.auth**, **Request.url**, **Request.header**, **Request.forms**, **Request.files**.
* The **Response.set\_cookie()** and **Request.get\_cookie()** methods are now able to encode and decode python objects. This is called a *secure cookie* because the encoded values are signed and protected from changes on client side. All pickle-able data structures are allowed.
* The new **Router** class drastically improves performance for setups with lots of dynamic routes and supports named routes (named route + dict = URL string).
* It is now possible (and recommended) to return [**HTTPError**](http://bottle.zzir.cn/index.html#bottle.HTTPError) and [**HTTPResponse**](http://bottle.zzir.cn/index.html#bottle.HTTPResponse) instances or other exception objects instead of raising them.
* The new function **static\_file()** equals **send\_file()** but returns a [**HTTPResponse**](http://bottle.zzir.cn/index.html#bottle.HTTPResponse) or [**HTTPError**](http://bottle.zzir.cn/index.html#bottle.HTTPError) instead of raising it. **send\_file()** is deprecated.
* New [**get()**](http://bottle.zzir.cn/index.html#bottle.get), [**post()**](http://bottle.zzir.cn/index.html#bottle.post), [**put()**](http://bottle.zzir.cn/index.html#bottle.put) and [**delete()**](http://bottle.zzir.cn/index.html#bottle.delete) decorators.
* The [**SimpleTemplate**](http://bottle.zzir.cn/index.html#bottle.SimpleTemplate) engine got full unicode support.
* Lots of non-critical bugfixes.

Contributors[¶](http://bottle.zzir.cn/index.html#contributors)

Bottle is written and maintained by Marcel Hellkamp <[marc@bottlepy.org](mailto:marc%40bottlepy.org)>.

Thanks to all the people who found bugs, sent patches, spread the word, helped each other on the mailing-list and made this project possible. I hope the following (alphabetically sorted) list is complete. If you miss your name on that list (or want your name removed) please [tell me](http://bottle.zzir.cn/index.html#document-contact) or add it yourself.

* acasajus
* Adam R. Smith
* Alexey Borzenkov
* Alexis Daboville
* Anton I. Sipos
* Anton Kolechkin
* apexi200sx
* apheage
* BillMa
* Brad Greenlee
* Brandon Gilmore
* Branko Vukelic
* Brian Sierakowski
* Brian Wickman
* Carl Scharenberg
* Damien Degois
* David Buxton
* Duane Johnson
* fcamel
* Frank Murphy
* Frederic Junod
* goldfaber3012
* Greg Milby
* gstein
* Ian Davis
* Itamar Nabriski
* Iuri de Silvio
* Jaimie Murdock
* Jeff Nichols
* Jeremy Kelley
* joegester
* Johannes Krampf
* Jonas Haag
* Joshua Roesslein
* Judson Neer
* Karl
* Kevin Zuber
* Kraken
* Kyle Fritz
* m35
* Marcos Neves
* masklinn
* Michael Labbe
* Michael Soulier
* [reddit](http://reddit.com/r/python)
* Nicolas Vanhoren
* Robert Rollins
* rogererens
* rwxrwx
* Santiago Gala
* Sean M. Collins
* Sebastian Wollrath
* Seth
* Sigurd Høgsbro
* Stuart Rackham
* Sun Ning
* Tomás A. Schertel
* Tristan Zajonc
* voltron
* Wieland Hoffmann
* zombat
* Thiago Avelino

开发者笔记[¶](http://bottle.zzir.cn/index.html#developer-notes)

这份文档是为那些对Bottle的开发和发布流程感兴趣的开发者和软件包维护者准备的。如果你想要做贡献，看它是对的！

参与进来[¶](http://bottle.zzir.cn/index.html#get-involved)

有多种加入社区的途径，保持消息的灵通。这里是一些方法:

* **邮件列表**: 发送邮件到 [bottlepy+subscribe@googlegroups.com](mailto:bottlepy+subscribe%40googlegroups.com) 以加入我们的邮件列表(无需Google账户)
* **Twitter**: [Follow us on Twitter](https://twitter.com/bottlepy) or search for the [#bottlepy](https://twitter.com/#!/search/%23bottlepy) tag.
* **IRC**: 加入 [#irc.freenode.net上的bottlepy频道](irc://irc.freenode.net/bottlepy) 或使用 [web聊天界面](http://webchat.freenode.net/?channels=bottlepy)
* **Google plus**: 有时我们会在Google+页面上发表一些 [与Bottle有关的博客](https://plus.google.com/b/104025895326575643538/104025895326575643538/posts)

获取源代码[¶](http://bottle.zzir.cn/index.html#get-the-sources)

Bottle的 [开发仓库](https://github.com/bottlepy/bottle) 和 [问题追踪](https://github.com/bottlepy/bottle/issues) 都搭建在 [github](https://github.com/bottlepy/bottle) 上面。如果你打算做贡献，创建一个github帐号然后fork一下吧。你做出的更改或建议都可被其他开发者看到，可以一起讨论。即使没有github帐号，你也可以clone代码仓库，或下载最新开发版本的压缩包。

* **git:** git clone git://github.com/bottlepy/bottle.git
* **git/https:** git clone https://github.com/bottlepy/bottle.git
* **Download:** Development branch as [tar archive](http://github.com/bottlepy/bottle/tarball/master) or [zip file](http://github.com/bottlepy/bottle/zipball/master).
* **Translations:** [transifex.com/projects/p/bottle](https://www.transifex.com/projects/p/bottle/)

发布和更新[¶](http://bottle.zzir.cn/index.html#releases-and-updates)

Bottle 通过 [PyPI](http://pypi.python.org/pypi/bottle)不定期发布。候选版本和过时版本的错误修订只能从git获取。有些Linux发行版可能提供过时版本。

Bottle的版本号分隔为三个部分(**major.minor.revision**)。版本号 *不会* 用于引入新特征，只是为了指出重要的bug-fix或API的改动。关键的bug会在近两个新的版本中修复，然后通过所有渠道发布(邮件列表, twitter, github)。不重要的bug修复或特征不保证添加到旧版本中。这个情况在未来也许会改变。

Major Release (x.0)

在重要的里程碑达成或新的更新和旧版本彻底不兼容时，会增加major版本号。为了使用新版本，你也许需要修改整个应用，但major版本号极少会改变。

Minor Release (x.y)

在更改了API之后，或增加minor版本号。你可能会收到一些API已过时的警告，调整设置以继续使用旧的特征，但在大多数情况下，这些更新都会对旧版本兼容。你应当保持更新，但这不是必须的。0.8版本是一个特例，它 *不兼容* 旧版本(所以我们跳过了0.7版本直接发布0.8版本)，不好意思。

Revision (x.y.z)

在修复了一些bug，和改动不会修改API的时候，会增加revision版本号。你可以放心更新，而不用修改你应用的代码。事实上，你确实应该更新这类版本，因为它常常修复一些安全问题。

Pre-Release Versions

RC版本会在版本号中添加 rc 字样。API已基本稳定，已经开放测试，但还没正式发布。你不应该在生产环境中使用rc版本。

代码仓库结构[¶](http://bottle.zzir.cn/index.html#repository-structure)

代码仓库的结构如下:

master branch

该分支用于集成，测试和开发。所有计划添加到下一版本的改动，会在这里合并和测试。

release-x.y branches

只要master分支已经可以用来发布一个新的版本，它会被安排到一个新的发行分支里面。在RC阶段，不再添加或删除特征，只接受bug-fix和小改动，直到认为它可以用于生产环境和正式发布。基于这点考虑，我们称之为“支持分支(support branch)”，依然接受bug-fix，但仅限关键的bug-fix。每次更改后，版本号都会更新，这样你就能及时获取重要的改动了。

bugfix\_name-x.y branches

这些分支是临时性的，用于修复现有发布版本的bug。在合并到其他分支后，它们就会被删除。

Feature branches

所有这类分支都是用于新增特征的。基于master分支，在开发、合并完毕后，就会被删除。

**对于开发者，这意味着什么？**

如果你想添加一个特征，可以从 master 分支创建一个分支。如果你想修复一个bug，可从 release-x.y 这类分支创建一个分支。无论是添加特征还是修复bug，都建议在一个独立的分支上进行，这样合并工作就简单了。就这些了！在页面底部会有git工作流程的例子。

Oh，请不要修改版本号。我们会在集成的时候进行修改。因为你不知道我们什么时候会将你的request pull下来:)

**对于软件包维护者，这意味着什么？**

关注那些bugfix和新版本的tag，还有邮件列表。如果你想从代码仓库中获取特定的版本，请使用tag，而不是分支。分支中也许会包含一些未发布的改动，但tag会标记是那个commit更改了版本号。

提交补丁[¶](http://bottle.zzir.cn/index.html#submitting-patches)

让你的补丁被集成进来的最好方法，是在github上面fork整个项目，创建一个新的分支，修改代码，然后发送一个pull-request。页面下方是git工作流程的例子，也许会有帮助。提交git兼容的补丁文件到邮件列表也是可以的。无论使用什么方法，请遵守以下的基本规则:

* **文档：** 告诉我们你的补丁做了什么。注释你的代码。如果你添加了新的特征，请添加相应的使用方法。
* **测试：** 编写测试以证明你的补丁如期工作，且没有破坏任何东西。如果你修复了一个bug，至少写一个测试用例来触发这个bug。在提交补丁之前，请确保所有测试已通过。
* **一次只提交一个补丁：** 一次只修改一个bug，一次只添加一个新特征。保持补丁的干净。
* **与上流同步：** 如果在你编写补丁的时候， upstream/master 分支被修改了，那么先rebase或将新的修改pull下来，确保你的补丁在合并的时候不会造成冲突。

生成文档[¶](http://bottle.zzir.cn/index.html#building-the-documentation)

你需要一个Sphinx的新版本来生产整份文档。建议将Sphinx安装到一个 **virtualenv** 环境。

*# Install prerequisites*

which virtualenv || sudo apt-get install python-virtualenv

virtualenv --no-site-dependencies venv

./venv/pip install -U sphinx

*# Clone or download bottle from github*

git clone https://github.com/bottlepy/bottle.git

*# Activate build environment*

source ./venv/bin/activate

*# Build HTML docs*

cd bottle/docs

make html

*# Optional: Install prerequisites for PDF generation*

sudo apt-get install texlive-latex-extra **\**

texlive-latex-recommended **\**

texlive-fonts-recommended

*# Optional: Build the documentation as PDF*

make latex

cd ../build/docs/latex

make pdf

GIT工作流程[¶](http://bottle.zzir.cn/index.html#git-workflow-examples)

接下来的例子都假设你已经有一个 [git的免费帐号](https://github.com/) 。虽然不是必须的，但可简单化很多东西。

首先，你需要从官方代码仓库创建一个fork。只需在 [bottle项目页面](https://github.com/bottlepy/bottle) 点击一下”fork”按钮就行了。创建玩fork之后，会得到一个关于这个新仓库的简介。

你刚刚创建的fork托管在github上面，对所有人都是可见的，但只有你有修改的权限。现在你需要将其从线上clone下面，做出实际的修改。确保你使用的是(可写-可读)的私有URL，而不是(只读)的公开URL。

git clone git**@github.com**:your\_github\_account/bottle.git

在你将代码仓库clone下来后，就有了一个”origin”分支，指向你在github上的fork。不要让名字迷惑了你，它并不指向bottle的官方代码仓库，只是指向你自己的fork。为了追踪官方的代码仓库，可添加一个新的”upstream”远程分支。

cd bottle

git remote add upstream git://github.com/bottlepy/bottle.git

git fetch upstream

注意，”upstream”分支使用的是公开的URL，是只读的。你不能直接往该分支push东西，而是由我们来你的公开代码仓库pull，后面会讲到。

**提交一个特征**

在独立的特征分支内开发新的特征，会令集成工作更简单。因为它们会被合并到 master 分支，所有它们必须是基于 upstream/master 的分支。下列命令创建一个特征分支。

git checkout -b cool\_feature upstream/master

现在可开始写代码，写测试，更新文档。在提交更改之前，记得确保所有测试已经通过。

git commit -a -m "Cool Feature"

与此同时，如果 upstream/master 这个分支有改动，那么你的提交就有可能造成冲突，可通过rebase操作来解决。

git fetch upstream

git rebase upstream

这相当于先撤销你的所有改动，更新你的分支到最新版本，再重做你的所有改动。如果你已经发布了你的分支(下一步会提及)，这就不是一个好主意了，因为会覆写你的提交历史。这种情况下，你应该先将最新版本pull下来，手动解决所有冲突，运行测试，再提交。

现在，你已经做好准备发一个pull-request了。但首先你应该公开你的特征分支，很简单，将其push到你github的fork上面就行了。

git push origin cool\_feature

在你push完你所有的commit之后，你需要告知我们这个新特征。一种办法是通过github发一个pull-request。另一种办法是把这个消息发到邮件列表，这也是我们推荐的方式，这样其他开发者就能看到和讨论你的补丁，你也能免费得到一些反馈 :)

如果我们接受了你的补丁，我们会将其集成到官方的开发分支中，它将成为下个发布版本的一部分。

**修复Bug**

修复Bug和添加一个特征差不多，下面是一些不同点:

1. 修复所有受影响的分支，而不仅仅是开发分支(Branch off of the affected release branches instead of just the development branch)。
2. 至少编写一个触发该Bug的测试用例。
3. 修复所有受影响的分支，包括 upstream/master ，如果它也受影响。 git cherry-pick 可帮你完成一些重复工作。
4. 字后面要加上其修复的版本号，以防冲突。例子: my\_bugfix-x.y 或 my\_bugfix-dev 。

插件开发指南[¶](http://bottle.zzir.cn/index.html#plugin-development-guide)

这份指南介绍了插件的API，以及如何编写自己的插件。我建议先阅读 [插件](http://bottle.zzir.cn/index.html#plugins) 这一部分，再看这份指南。 [可用插件列表](http://bottle.zzir.cn/index.html#document-plugins/index) 这里也有一些实际的例子。

注解

这是一份初稿。如果你发现了任何错误，或某些部分解释的不够清楚，请通过 [邮件列表](mailto:bottlepy%40googlegroups.com) 或 [bug report](https://github.com/bottlepy/bottle/issues) 告知。

插件工作方式：基础知识[¶](http://bottle.zzir.cn/index.html#how-plugins-work-the-basics)

插件的API是通过Python的 [修饰器](http://docs.python.org/glossary.html#term-decorator) 来实现的。简单来说，一个插件就是应用在route回调函数上的修饰器。

这只是一个简单的例子。插件还可以做更多路由回调，但这是一个良好的起点。让我们来看看代码：

**from** **bottle** **import** response, install

**import** **time**

**def** stopwatch(callback):

**def** wrapper(\*args, \*\*kwargs):

start = time.time()

body = callback(\*args, \*\*kwargs)

end = time.time()

response.headers['X-Exec-Time'] = str(end - start)

**return** body

**return** wrapper

install(stopwatch)

这个插件计算每次请求的响应时间，并在响应头中添加了 X-Exec-Time 字段。如你所见，插件返回了一个wrapper函数，由它来调用原先的回调函数。这就是修饰器的常见工作方式了。

最后一行，将该插件安装到Bottle的默认应用里面。这样，应用中的所有route都会应用这个插件了。就是说，每次请求都会调用 stopwatch() ，更改了route的默认行为。

插件是按需加载的，就是在route第一次被访问的时候加载。为了在多线程环境下工作，插件应该是线程安全的。在大多数情况下，这都不是一个问题，但务必提高警惕。

一旦route中使用了插件后，插件中的回调函数会被缓存起来，接下来都是直接使用缓存中的版本来响应请求。意味着每个route只会请求一次插件。在应用的插件列表变化的时候，这个缓存会被清空。你的插件应当可以多次修饰同一个route。

这种修饰器般的API受到种种限制。你不知道route或相应的应用对象是如何被修饰的，也不知道如何有效地存储那些在route之间共享的数据。但别怕！插件不仅仅是修饰器函数。只要一个插件是callable的或实现了一个扩展的API(后面会讲到)，Bottle都可接受。扩展的API给你更多的控制权。

插件API[¶](http://bottle.zzir.cn/index.html#plugin-api)

Plugin 类不是一个真正的类(你不能从bottle中导入它)，它只是一个插件需要实现的接口。只要一个对象实现了以下接口，Bottle就认可它作为一个插件。

*class* **Plugin**(*object*)[¶](http://bottle.zzir.cn/index.html#bottle.Plugin)

插件应该是callable的，或实现了 [**apply()**](http://bottle.zzir.cn/index.html#bottle.Plugin.apply) 方法。如果定义了 [**apply()**](http://bottle.zzir.cn/index.html#bottle.Plugin.apply) 方法，那么会优先调用，而不是直接调用插件。其它的方法和属性都是可选的。

**name**[¶](http://bottle.zzir.cn/index.html#bottle.Plugin.name)

[**Bottle.uninstall()**](http://bottle.zzir.cn/index.html#bottle.Bottle.uninstall) 方法和 [**Bottle.route()**](http://bottle.zzir.cn/index.html#bottle.Bottle.route) 中的 *skip* 参数都接受一个与名字有关的字符串，对应插件或其类型。只有插件中有一个name属性的时候，这才会起作用。

**api**[¶](http://bottle.zzir.cn/index.html#bottle.Plugin.api)

插件的API还在逐步改进。这个整形数告诉Bottle使用哪个版本的插件。如果没有这个属性，Bottle默认使用第一个版本。当前版本是 2 。详见 [插件API的改动](http://bottle.zzir.cn/index.html#plugin-changelog) 。

**setup**(*self*, *app*)[¶](http://bottle.zzir.cn/index.html#bottle.Plugin.setup)

插件被安装的时候调用(见 [**Bottle.install()**](http://bottle.zzir.cn/index.html#bottle.Bottle.install) )。唯一的参数是相应的应用对象。

**\_\_call\_\_**(*self*, *callback*)[¶](http://bottle.zzir.cn/index.html#bottle.Plugin.__call__)

如果没有定义 [**apply()**](http://bottle.zzir.cn/index.html#bottle.Plugin.apply) 方法，插件本身会被直接当成一个修饰器使用(译者注：Python的Magic Method，调用一个类即是调用类的\_\_call\_\_函数)，应用到各个route。唯一的参数就是其所修饰的函数。这个方法返回的东西会直接替换掉原先的回调函数。如果无需如此，则直接返回未修改过的回调函数即可。

**apply**(*self*, *callback*, *route*)[¶](http://bottle.zzir.cn/index.html#bottle.Plugin.apply)

如果存在，会优先调用，而不调用 [**\_\_call\_\_()**](http://bottle.zzir.cn/index.html#bottle.Plugin.__call__) 。额外的 *route* 参数是 [**Route**](http://bottle.zzir.cn/index.html#bottle.Route) 类的一个实例，提供很多该route信息和上下文。详见 [Route上下文](http://bottle.zzir.cn/index.html#route-context) 。

**close**(*self*)[¶](http://bottle.zzir.cn/index.html#bottle.Plugin.close)

插件被卸载或应用关闭的时候被调用，详见 [**Bottle.uninstall()**](http://bottle.zzir.cn/index.html#bottle.Bottle.uninstall) 或 [**Bottle.close()**](http://bottle.zzir.cn/index.html#bottle.Bottle.close) 。

[**Plugin.setup()**](http://bottle.zzir.cn/index.html#bottle.Plugin.setup) 方法和 [**Plugin.close()**](http://bottle.zzir.cn/index.html#bottle.Plugin.close) 方法 *不* 会被调用，如果插件是通过 [**Bottle.route()**](http://bottle.zzir.cn/index.html#bottle.Bottle.route) 方法来应用到route上面的，但会在安装插件的时候被调用。

插件API的改动[¶](http://bottle.zzir.cn/index.html#plugin-api-changes)

插件的API还在不断改进中。在Bottle 0.10版本中的改动，定位了route上下文字典中已确定的问题。为了保持对0.9版本插件的兼容，我们添加了一个可选的 [**Plugin.api**](http://bottle.zzir.cn/index.html#bottle.Plugin.api) 属性，告诉Bottle使用哪个版本的API。API之间的不同点总结如下。

* **Bottle 0.9 API 1** (无 [**Plugin.api**](http://bottle.zzir.cn/index.html#bottle.Plugin.api) 属性)
  + Original Plugin API as described in the 0.9 docs.
* **Bottle 0.10 API 2** ( [**Plugin.api**](http://bottle.zzir.cn/index.html#bottle.Plugin.api) 属性为2)
  + The *context* parameter of the [**Plugin.apply()**](http://bottle.zzir.cn/index.html#bottle.Plugin.apply) method is now an instance of [**Route**](http://bottle.zzir.cn/index.html#bottle.Route) instead of a context dictionary.

Route上下文[¶](http://bottle.zzir.cn/index.html#the-route-context)

[**Route**](http://bottle.zzir.cn/index.html#bottle.Route) 的实例被传递给 [**Plugin.apply()**](http://bottle.zzir.cn/index.html#bottle.Plugin.apply) 函数，以提供更多该route的相关信息。最重要的属性总结如下。

| **属性** | **描述** |
| --- | --- |
| app | 安装该route的应用对象 |
| rule | route规则的字符串 (例如： /wiki/:page) |
| method | HTTP方法的字符串(例如： GET) |
| callback | 未应用任何插件的原始回调函数，用于内省。 |
| name | route的名字，如未指定则为 None |
| plugins | route安装的插件列表，除了整个应用范围内的插件，额外添加的(见 [**Bottle.route()**](http://bottle.zzir.cn/index.html#bottle.Bottle.route) ) |
| skiplist | 应用安装了，但该route没安装的插件列表(见 meth:*Bottle.route* ) |
| config | 传递给 [**Bottle.route()**](http://bottle.zzir.cn/index.html#bottle.Bottle.route) 修饰器的额外参数，存在一个字典中，用于特定的设置和元数据 |

对你的应用而言， [**Route.config**](http://bottle.zzir.cn/index.html#bottle.Route.config) 也许是最重要的属性了。记住，这个字典会在所有插件中共享，建议添加一个独一无二的前缀。如果你的插件需要很多设置，将其保存在 *config* 字典的一个独立的命名空间吧。防止插件之间的命名冲突。

改变 [**Route**](http://bottle.zzir.cn/index.html#bottle.Route) 对象[¶](http://bottle.zzir.cn/index.html#changing-the-route-object)

[**Route**](http://bottle.zzir.cn/index.html#bottle.Route) 的一些属性是不可变的，改动也许会影响到其它插件。坏主意就是，monkey-patch一个损坏的route，而不是提供有效的帮助信息来让用户修复问题。

在极少情况下，破坏规则也许是恰当的。在你更改了 [**Route**](http://bottle.zzir.cn/index.html#bottle.Route) 实例后，抛一个 **RouteReset** 异常。这会从缓存中删除当前的route，并重新应用所有插件。无论如何，router没有被更新。改变 *rule* 或 *method* 的值并不会影响到router，只会影响到插件。这个情况在将来也许会改变。

运行时优化[¶](http://bottle.zzir.cn/index.html#runtime-optimizations)

插件应用到route以后，被插件封装起来的回调函数会被缓存，以加速后续的访问。如果你的插件的行为依赖一些设置，你需要在运行时更改这些设置，你需要在每次请求的时候读取设置信息。够简单了吧。

然而，为了性能考虑，也许值得根据当前需求，选择一个不同的封装，通过闭包，或在运行时使用、禁用一个插件。让我们拿内置的HooksPlugin作为一个例子(译者注：可在bottle.py搜索该实现)：如果没有安装任何钩子，这个插件会从所有受影响的route中删除自身，不做任何工作。一旦你安装了第一个钩子，这个插件就会激活自身，再次工作。

为了达到这个目的，你需要控制回调函数的缓存： [**Route.reset()**](http://bottle.zzir.cn/index.html#bottle.Route.reset) 函数清空单一route的缓存， [**Bottle.reset()**](http://bottle.zzir.cn/index.html#bottle.Bottle.reset) 函数清空所有route的缓存。在下一次请求的时候，所有插件被重新应用到route上面，就像第一次请求时那样。

如果在route的回调函数里面调用，两种方法都不会影响当前的请求。当然，可以抛出一个 **RouteReset** 异常，来改变当前的请求。

插件例子： SQLitePlugin[¶](http://bottle.zzir.cn/index.html#plugin-example-sqliteplugin)

这个插件提供对sqlite3数据库的访问，如果route的回调函数提供了关键字参数(默认是”db”)，则”db”可做为数据库连接，如果route的回调函数没有提供该参数，则忽略该route。wrapper不会影响返回值，但是会处理插件相关的异常。 [**Plugin.setup()**](http://bottle.zzir.cn/index.html#bottle.Plugin.setup) 方法用于检查应用，查找冲突的插件。

**import** **sqlite3**

**import** **inspect**

**class** **SQLitePlugin**(object):

*''' This plugin passes an sqlite3 database handle to route callbacks*

*that accept a 'db' keyword argument. If a callback does not expect*

*such a parameter, no connection is made. You can override the database*

*settings on a per-route basis. '''*

name = 'sqlite'

api = 2

**def** \_\_init\_\_(self, dbfile=':memory:', autocommit=True, dictrows=True,

keyword='db'):

self.dbfile = dbfile

self.autocommit = autocommit

self.dictrows = dictrows

self.keyword = keyword

**def** setup(self, app):

*''' Make sure that other installed plugins don't affect the same*

*keyword argument.'''*

**for** other **in** app.plugins:

**if** **not** isinstance(other, SQLitePlugin): **continue**

**if** other.keyword == self.keyword:

**raise** PluginError("Found another sqlite plugin with "\

"conflicting settings (non-unique keyword).")

**def** apply(self, callback, context):

*# Override global configuration with route-specific values.*

conf = context.config.get('sqlite') **or** {}

dbfile = conf.get('dbfile', self.dbfile)

autocommit = conf.get('autocommit', self.autocommit)

dictrows = conf.get('dictrows', self.dictrows)

keyword = conf.get('keyword', self.keyword)

*# Test if the original callback accepts a 'db' keyword.*

*# Ignore it if it does not need a database handle.*

args = inspect.getargspec(context.callback)[0]

**if** keyword **not** **in** args:

**return** callback

**def** wrapper(\*args, \*\*kwargs):

*# Connect to the database*

db = sqlite3.connect(dbfile)

*# This enables column access by name: row['column\_name']*

**if** dictrows: db.row\_factory = sqlite3.Row

*# Add the connection handle as a keyword argument.*

kwargs[keyword] = db

**try**:

rv = callback(\*args, \*\*kwargs)

**if** autocommit: db.commit()

**except** sqlite3.IntegrityError, e:

db.rollback()

**raise** HTTPError(500, "Database Error", e)

**finally**:

db.close()

**return** rv

*# Replace the route callback with the wrapped one.*

**return** wrapper

这个插件十分有用，已经和Bottle提供的那个版本很类似了(译者注：=。= 一模一样)。只要60行代码，还不赖嘛！下面是一个使用例子。

sqlite = SQLitePlugin(dbfile='/tmp/test.db')

bottle.install(sqlite)

**@route**('/show/:page')

**def** show(page, db):

row = db.execute('SELECT \* from pages where name=?', page).fetchone()

**if** row:

**return** template('showpage', page=row)

**return** HTTPError(404, "Page not found")

**@route**('/static/:fname#.\*#')

**def** static(fname):

**return** static\_file(fname, root='/some/path')

**@route**('/admin/set/:db#[a-zA-Z]+#', skip=[sqlite])

**def** change\_dbfile(db):

sqlite.dbfile = '/tmp/*%s*.db' % db

**return** "Switched DB to *%s*.db" % db

第一个route提供了一个”db”参数，告诉插件它需要一个数据库连接。第二个route不需要一个数据库连接，所以会被插件忽略。第三个route确实有一个”db”参数，但显式的禁用了sqlite插件，这样，”db”参数不会被插件修改，还是包含URL传过来的那个值。

Contact[¶](http://bottle.zzir.cn/index.html#contact)

**关于作者**

Hi, I’m *Marcel Hellkamp* (aka *defnull*), author of Bottle and the guy behind this website. I’m 27 years old and studying computer science at the Georg-August-University in Göttingen, Germany. Python is my favorite language, but I also code in ruby and JavaScript a lot. Watch me on [twitter](http://twitter.com/bottlepy) or visit my profile at [GitHub](http://github.com/defnull) to get in contact. A [mailinglist](http://groups.google.de/group/bottlepy) is open for Bottle related questions, too.

**关于Bottle**

This is my first open source project so far. It started and a small experiment but soon got so much positive feedback I decided to make something real out of it. Here it is.

**Impressum und Kontaktdaten**

(This is required by [German law](http://bundesrecht.juris.de/tmg/__5.html))

Die Nutzung der folgenden Kontaktdaten ist ausschließlich für die Kontaktaufnahme mit dem Betreiber dieser Webseite bei rechtlichen Problemen vorgesehen. Insbesondere die Nutzung zu Werbe- oder ähnlichen Zwecken ist ausdrücklich untersagt.

* **Betreiber**: Marcel Hellkamp
* **Ort**: D - 37075 Göttingen
* **Strasse**: Theodor-Heuss Strasse 13
* **Telefon**: +49 (0) 551 20005915
* **E-Mail**: marc at gsites dot de

许可证[¶](http://bottle.zzir.cn/index.html#license)

代码和文件皆使用MIT许可证:

Copyright (c) 2015, Marcel Hellkamp.

Permission **is** hereby granted, free of charge, to any person obtaining a copy

of this software **and** associated documentation files (the "Software"), to deal

**in** the Software without restriction, including without limitation the rights

to use, copy, modify, merge, publish, distribute, sublicense, **and**/**or** sell

copies of the Software, **and** to permit persons to whom the Software **is**

furnished to do so, subject to the following conditions:

The above copyright notice **and** this permission notice shall be included **in**

all copies **or** substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR

IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,

FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE

AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER

LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,

OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN

THE SOFTWARE.

然而，许可证 *不包含* Bottle的logo。logo用作指向Bottle主页的连接，或未修改过的类库。如要用于其它用途，请先请求许可。

**脚注**

|  |  |
| --- | --- |
| [[1]](http://bottle.zzir.cn/index.html#id2) | Usage of the template or server adapter classes requires the corresponding template or server modules. |