

Google App Engine入门:简介

(本文译自:[Google App Engine Getting Started](#))

欢迎使用 Google App Engine! 创建一个Google App Engine应用非常简便, 只需要花费你几分钟时间. 你可以很方便的创建你的网站应用: 直接上传并分享, 不需要进行任何的修改和注释。

在这个指导里, 我们将创建一个简单的留言本, 让用户可以发表留言. 并且支持匿名和Google帐号两种方式发表留言。

这个留言本程序将演示如何使用Google的数据存储, 集成Google账户系统, 如何使用一个简单的Python Web framework来调试GAE网站. 并且还将演示如何使用Django的模版引擎。

接下来...

为了开始编写你的Google App Engine应用程序, 你必须先下载 Google App Engine软件开发环境。

Google App Engine 入门:开发环境

(本文译自:[Google App Engine Getting Started](#))

Google 提供了一个Google App Engine 软件开发包 (SDK), 用于让开发者进行网站应用程序开发或上传已经完成的应用。

这个开发包包含:

- 一个web服务程序, 用来模拟App Engine应用环境

- 一个本地版的数据存储方案

- 本地模拟的Google帐号集成

- 支持使用Api来分析URL和发送邮件

这个开发包可以运行在所有安装了Python2.5的机器上, 并且支持Windows, Mac OS X 和 Linux系统。

因为这个开发包是以Python2.5为基础的, 所以你必须先要在你的机器上安装Python2.5(必须是2.5版本). Mac OS X 10.5 用户可能已经安装了Python2.5, 某些版本的Linux自带Python环境, 但是版本可能比较低, 需要升级成2.5版本。

接下来: [下载并安装App Engine SDK](#)

在这个入门教程中, 我们需要用到下面的两个命令:

- `dev_appserver.py`, 本地开发服务程序

- `appcfg.py`, 上传并发布你的应用

Windows或Mac OS X的安装程序会将这两个命令放到命令行运行路径中. 在安装结束后, 你可以直接通过命令行执行。

如果你使用的是Zip压缩包格式的SDK,你可以google_appengine目录下找到它们。

接下来...

本地开发环境,让你可以轻松的开发和测试你的应用。并且保持和发布后的环境完全的一致。下面,让我们开始编写代码吧:

Google App Engine 入门: Hello World

(本文译自:[Google App Engine Getting Started](#))

Google App Engine 应用通过 [CGI](#) 标准协议与服务器通讯.这是一个标准的Http处理流程,Web服务接受到客户端发来的Get或Post请求,web服务器把请求转发给你的应用程序,由应用程序来处理要输出的内容。

为了更好的理解这个过程,下面就开始开发我们经典的Hello World应用程序吧。在这一章,仅仅只是实现显示一些简单的信息的功能。

创建一个简单的 Request Handler

首先创建一个名为 helloworld 的文件夹。除非特殊说明,以后所有关于这个应用程序的文件都将放在这个文件夹里面。

在helloworld 文件夹里,创建一个新文件 helloworld.py,文件内容如下:

```
class="prettyprint"print 'Content-Type: text/plain'
print ''
print 'Hello, world!'
```

这个Python 脚本处理一个request请求,并且设置一个Http header,输出一个空行和一段信息Hello, world!.

创建配置文件

每个App Engine application 都包含一个名为 app.yaml的配置文件。在这个配置文件中,可以设置具体的某个URL需要用哪个Python脚本来处理.

现在,在 helloworld 文件夹中,创建一个新的 app.yaml 文件,输入以下内容:

```
application: helloworld
version: 1
runtime: python
api_version: 1

handlers:
```

- url: /*

script: helloworld.py

这个配置文件描述了以下内容：

这个应用程序的标识是 `helloworld`。这个标识需要和你在App Engine网站上创建的应用程序标识保持一致。在开发期间你可以使用任何你喜欢的名字，但是上传的时候，必须要和你在App Engine 注册的标识保持一致。现在，我们把它设置为 `helloworld`。

你的应用程序的版本号为1，如果你在上传应用之前修改了这个编号，App Engine 将会自动保留前一个版本的副本，以方便你可以在管理平台中将当前版本恢复成原来的版本。

该应用运行在 `python` 环境，环境版本是 1。目前只有Python可选，将来会提供更多的运行环境和开发语言。

所有符合正则表达式/* (所有URL) 的请求，都由 `helloworld.py` 脚本来处理。

该配置文件使用 [YAML](#)语法。关于该配置文件的更多选项，请参考 [the app.yaml reference](#)。

测试应用程序

现在这个应用程序已经基本上完整了。你可以在本地App Engine SDK环境中进行模拟运行测试。

首先，指定应用路径为 `helloworld` 目录，使用下面的命令启动测试环境Web服务程序：

```
google_appengine/dev_appserver.py helloworld/
```

这个Web服务程序将监听8080端口。你可以在浏览器中输入以下地址进行测试：

<http://localhost:8080/>

关于这个web服务程序的更多选项（如怎样修改默认端口等），请查看[the Dev Web Server reference](#)，或者使用命令行选项 `--help`。

无需中断你的开发

在开发过程中，你不需要不停的重启Web服务程序。Web服务程序可以自行判断哪些脚本文件已经被修改过了，并且重新加载这些脚本。

试一试：不要关闭web服务程序，编辑 `helloworld.py` 将 `Hello, world!` 修改成其他内容。重新访问<http://localhost:8080/>，看看是不是您的修改已经生效了！

要关闭Web服务程序，您只需在控制台中按下 `Control-C` (或其他有效的 "break" 功能键)。

PS：在以后的入门指导中，你可以让你的Web服务程序一直开着。

接下来...

我们已经开发了一个完整的web应用程序！你可能迫不及待的就想把这个程序发布出去，并分享给你的朋友了。且慢，还是让我们给它增加一些功能后再这样做吧，毕竟它太简陋了。

Google App Engine 入门：使用webapp framework

(本文译自:[Google App Engine Getting Started](#))

使用CGI模式来开发网页是相当简单的,但是如果这样手工完成所有的代码无疑是相当让人感到苦恼的事情。那么,使用Web application frameworks来处理具体的细节是个不错的选择, Google App Engine supports 支持所有基于python的CGI, WCGI框架应用,包括 [Django](#), [CherryPy](#), [Pylons](#), [web.py](#)等等... 你可以选择其中的任何一个进行开发。(就我看来, GAE整个框架多处参考了Django, 所以使用Django框架,是个不错的选择)。

GAE内置了一个叫做webapp的WEB应用框架,并包含在SDK环境中,你可以直接在你的应用中使用它。接下来,我们就使用这个框架来完成后面的教程。

Hello, webapp!

一个 webapp 应用包含三部分

- 一个或多个 RequestHandler 类用来处理http请求和应答

- 一个 WSGIApplication 实例,根据不同的URL请求,将处理交给不同的 RequestHandler 类实例。

- 一个主过程,通过CGI adaptor方式运行 WSGIApplication

让我们用webapp 将helloworld的例子重新修改一下吧。

编辑 helloworld/helloworld.py 并且把内容修改为:

```
class="prettyprint"import wsgiref.handlers

from google.appengine.ext import webapp

class MainPage(webapp.RequestHandler):
    def get(self):
        self.response.headers['Content-Type'] = 'text/plain'
        self.response.out.write('Hello, webapp World!')

def main():
    application = webapp.WSGIApplication(
        [('/', MainPage)],
        debug=True)
    wsgiref.handlers.CGIHandler().run(application)

if __name__ == "__main__":
    main()
```

刷新<http://localhost:8080/> 查看输出结果,是不是还是很简单啊。

What webapp Does

webapp 模块可以在 `google.appengine.ext` 包中找到。这个模块由GAE SDK提供，并且和实际运行环境是一致的。

上面的代码声明了一个 `request handler`, `MainPage`, 把 `URL(/)` 交给应用程序来处理。当 `webapp` 接受到一个形如 `/` 这样 `HTTP GET` 请求，就会调用 `MainPage` 类，并且调用 `get` 请求。在这个方面里面，关于请求的所有信息可以使用 `self.request` 来调用，一般来说，我们会在这里通过 `self.response` 设置应答内容，然后退出，`webapp` 会把应答的内容返回给 `MainPage` 实例。

这个应用程序本身是一个 `webapp.WSGIApplication`，我们可以设置参数 `debug=true`，让 `webapp` 在遇到错误时在浏览器中显示堆栈调用信息。但是，在发布前，请记得要把这个选项去掉。=

这段代码中使用了 `wsgiref` 模块，这个模块提供基本的CGI处理功能，包含在Python标准库中。要了解关于这么模块的更多信息，请查看 [the wsgiref module documentation](#)。

我们将会在后面的教程中使用更多 `webapp` 的功能。想要知道 `webapp` 更详细信息，请查看 [the webapp reference](#)。

接下来...

WEB应用程序框架使GAE开发更加简单，快速，并且减少了错误的发生。但 `webapp` 只是Python的WEB应用框架之一。接下来，我们会逐步完善它，并为它增加一些功能。

Google App Engine 入门:用户系统服务

Google App Engine 在其SDK中提供了很多有用的服务。其中之一就是用户系统服务，这个服务让我们可以轻松的集成Google的用户帐号系统，这样使用我们的应用的用户，就不需要注册单独的帐号，直接使用现有的Google帐号就可以登录了。

下面，我们为Hello word加入用户判断功能。

使用Users

重新编辑 `helloworld/helloworld.py`, 将代码修改如下:

```
class="prettyprint"import wsgiref.handlers

from google.appengine.api import users
from google.appengine.ext import webapp

class MainPage(webapp.RequestHandler):
    def get(self):
        user = users.get_current_user()

        if user:
            self.response.headers['Content-Type'] = 'text/plain'
            self.response.out.write('Hello, ' + user.nickname())
```

```

else:
    self.redirect(users.create_login_url(self.request.uri))

def main():
    application = webapp.WSGIApplication(
        [('/', MainPage)],
        debug=True)
    wsgiref.handlers.CGIHandler().run(application)

if __name__ == "__main__":
    main()

```

刷新页面，你会发现直接跳转到了本地模拟的Google登录页面了，在这个页面输入你的用户名，你的web应用程序就会为你生成一个 User 对象。

当你发布你的应用之后，这个程序将会直接跳转到真正的Google登录页面，等你登录成功后再跳转到你的程序页面。

The Users API

我们仔细查看一下代码:

```
class="prettyprint" user = users.get_current_user()
```

如果用户已经登录, `get_current_user()` 返回 User 对象, 否则返回 None.

```
class="prettyprint" if user:
```

```

    self.response.headers['Content-Type'] = 'text/plain'
    self.response.out.write('Hello, ' + user.nickname())

```

如果用户已经登录, 向登录用户显示一段欢迎信息

```
class="prettyprint" else:
```

```

    self.redirect(users.create_login_url(self.request.uri))

```

如果用户尚未登录, 调用 `webapp` 跳转到Google 帐号登录页面。这个跳转页面包含了登录成功后要返回的URL (`self.request.uri`), 用户登录成功后, 就会回到指定的页面了。

想了解更多信息, 请查看 [the Users reference](#).

接下来...

我们的应用程序, 已经实现了可以显示登录者的名字了, 接下来我们将为它增加留言的功能。

Google App Engine 入门 :处理表单数据

如果我们希望能显示用户自己的留言, 就需要用户通过web表单提交一些信息, 然后GAE程序对它们进行处理。Webapp框架使得实现这个过程相当简单。

使用webapp处理表单数据

重新修改 helloworld/helloworld.py :

```
style="background-color: White;"
```

```
import cgi
```

```
import wsgiref.handlers
```

```
from google.appengine.api import users
```

```
from google.appengine.ext import webapp
```

```
class MainPage(webapp.RequestHandler):
```

```
    def get(self):
```

```
        self.response.out.write("""
```

```
            <html>
```

```
                <body>
```

```
                    <form action="/sign" method="post">
```

```
                        <div><textarea name="content" rows="3" cols="60"></textarea></div>
```

```
                        <div><input type="submit" value="Sign Guestbook"></div>
```

```
                    </form>
```

```
                </body>
```

```
            </html>""")
```

```
class Guestbook(webapp.RequestHandler):
```

```
    def post(self):
```

```
        self.response.out.write('<html><body>You wrote:<pre>')
```

```
        self.response.out.write(cgi.escape(self.request.get('content')))
```

```
        self.response.out.write('</pre></body></html>')
```

```
def main():
```

```
    application = webapp.WSGIApplication(
```

```
        [('/', MainPage),
```

```
         ('/sign', Guestbook)],
```

```
        debug=True)
```

```
wsgiref.handlers.CGIHandler().run(application)
```

```
if __name__ == "__main__":  
    main()
```

这个页面显示一个输入框，并试图提交一些信息。

这个版本有两个处理过程: `MainPage`, 对应 URL `/`, 显示 web 表单. `Guestbook`, 对应 URL `/sign`, 显示刚刚提交的信息 .

`Guestbook handler` 实现了一个`post()`方法. 如果你需要同时支持 `GET` 和 `POST` 方法, 你可以同时定义这两个方法 .

代码中的`post()` 方法通过`self.request`获取用户提交的数据, 并在返回显示结果之前, 用`cgi.escape()`函数对一些HTML中的特殊字符(如`<>`)进行了处理. `cgi`是一个Python标准模块, 查看 [the documentation for cgi](#) 了解更多相关信息.

Note: App Engine 环境包含了Python2.5中的所有标准库. 但是, 不是所有方法都是允许的. 为了安全性, GAE程序运载在一个受限制的运行环境之中. 例如, 涉及操作系统底层的函数, 网络操作, 一部门文件操作是不被允许的. 如果, 你试图调用的话, 会提示一个错误. 想了解更详细的情况, 请查看 [The Python Runtime Environment](#).

接下来...

现在, 我们已经可以获取用户提交的数据了, 接下来, 我们还需要一种方式, 把这些信息保存起来.

Google App Engine 入门:数据存储

数据存储是个复杂的问题. 用户可能在一个特定的时间发出了一个数据请求, 但是下一个时间又发出了另外一个完全不同的数据请求. 所有的WEB服务都需要协调这些相互影响的请求, 并且这些请求可能来自世界的各个地方.

感谢Google App Engine,您再也不需要为这些发愁了. Google App Engine架构将为您处理所有关于数据分发, 负载均衡的问题, 并且提供了API来实现所有关于数据存储的问题.

存储已经提交的留言

App Engine 包含了一个基于Python的数据存储模型. 这个模型类似于 [Django's data modelling API](#), 但是使用了Google自己的存储环境.

对于上一章实现的留言程序,我们想要把用户提交的留言保存起来, 每个留言都包含作者名称, 消息内容, 发布时间等等, 并且按照留言的先后将其显示出来.

编辑 `helloworld/helloworld.py`, 在顶部添加 `import`代码 :

```
class="prettyprint"from google.appengine.ext import db
```

添加 Greeting类:

```
class="prettyprint"class Greeting(db.Model):  
    author = db.UserProperty()  
    content = db.StringProperty(multiline=True)  
    date = db.DateTimeProperty(auto_now_add=True)
```

上面的代码定义了一个数据模型类Greeting，这个类包含三个属性: author -User object类型, content - 字符串类型, date -日期类型.

其中一些属性包含了默认值:比如db.StringProperty 类型中 multiline=True 表明该字符串中可以包含换行符;db.DateTimeProperty类型中 auto_now_add=True 表明当Greeting对象创建的时候，将使用当前时间初始化这个属性。关于数据模型的属性的更多帮助，请查看 [the Datastore reference](#).

现在我们已经定义了一个数据对象模型，接下来，我们创建一个Greeting对象，并且把它保存起来。编辑 Guestbook handler :

```
class="prettyprint"class Guestbook(webapp.RequestHandler):  
    def post(self):  
        greeting = Greeting()  
  
        if users.get_current_user():  
            greeting.author = users.get_current_user()  
  
        greeting.content = self.request.get('content')  
        greeting.put()  
        self.redirect('/')
```

Guestbook 处理过程创建了一个Greeting 对象，设置了它的属性，然后使用 greeting.put()方法把它保存了起来。如果对象已经存在，使用put()方法，将会更新原先已经存在的数据。在这段代码里，我们是直接创建了Greeting对象，所以put()方法将把一个新的Greeting对象保存起来。

通过GQL查询数据

App Engine datastore 使用了一套复杂的数据储存系统.但是它并不是一个标准的关系数据库，所以不能使用标准的Sql语句进行查询。作为一个替代，Google准备了一套类 Sql的查询语句，称之为GQL.GQL 提供了和SQL基本类似的语法来读取数据.

编辑MainPage :

```
style="background-color: White;"  
  
class MainPage(webapp.RequestHandler):
```

```

def get(self):
    self.response.out.write('<html><body>')

    greetings = db.GqlQuery("SELECT * FROM Greeting ORDER BY date DESC LIMIT 10")

    for greeting in greetings:
        if greeting.author:
            self.response.out.write('<b>%s</b> wrote: ' % greeting.author.nickname())
        else:
            self.response.out.write('An anonymous person wrote:')
            self.response.out.write('<blockquote>%s</blockquote>' %
                                    cgi.escape(greeting.content))

    # Write the submission form and the footer of the page
    self.response.out.write("""
        <form action="/sign" method="post">
            <div><textarea name="content" rows="3" cols="60"></textarea></div>
            <div><input type="submit" value="Sign Guestbook"></div>
        </form>
    </body>
    </html>""")

```

刷新<http://localhost:8080/>，您可以提交一个留言看一下效果。

以下是相关的查询语句：

```

class="prettyprint"      greetings = db.GqlQuery("SELECT * FROM Greeting ORDER BY date
DESC LIMIT 10")

```

你也可以使用Greeting类的gql(...)方法来进行查询，使用这种方式时可以省略掉 SELECT * FROM Greeting :

```

class="prettyprint"      greetings = Greeting.gql("ORDER BY date DESC LIMIT 10")

```

和SQL语句一样，关键字（如：SELECT）是大小写不敏感的，但是字段名是大小写敏感的。

要注意的是，GQL语句总是返回完整的对象，所以GQL查询语句不能指定要查询的字段名。

也就是说，所有的GQL语句都是以SELECT * FROM *model* 开头的。

一个GQL查询语句可以用 WHERE指定查询条件，你可以指定一个或多个条件。和SQL不同的是，GQL查询不能包含变量值：GQL使用参数绑定查询中所有的变量。例如，获取当前登录用户的留言：

```
class="prettyprint"    if users.get_current_user():
    greetings = Greeting.gql("WHERE author = :1 ORDER BY date DESC",
                             users.get_current_user())
```

你也可以使用命名参数：

```
class="prettyprint"    greetings = Greeting.gql("WHERE author = :author ORDER BY date
DESC",
                                     author=users.get_current_user())
```

另外，Google datastore API 还提供了另外一种获取数据的方法：

```
class="prettyprint"    greetings = Greeting.all()
    greetings.filter("author =", users.get_current_user())
    greetings.order("-date")
```

想了解 GQL查询语法的更多内容，请查看 [the Datastore reference](#)。

清空开发环境的存储数据

为了方便你测试自己的应用，GAE开发环境使用了一个临时文件来保存本地的数据，要清空本地开发环境的数据，可以使用如下的命令行：

```
dev_appserver.py --clear_datastore helloworld/
```

接下来...

现在我们已经创建一个完整的留言板，但是这个版本中的HTML代码都包含在MainPage处理过程中，如果我们想要修改一下WEB应用的外观，将是一件麻烦的事情，接下来我们将学习如何使用模板，添加图片，CSS等静态文件。

Google App Engine 入门:使用模版文件

(本文译自:[Google App Engine Getting Started](#))

HTML代码是非常杂乱的，所以我们最好使用一个独立的文件来专门处理HTML代码，以便于将界面显示和数据获取的过程相互独立出来。有很多使用Python实现的模板系统，比如：

[EZT](#), [Cheetah](#), [ClearSilver](#), [Quixote](#), [Django](#) 等等.你可以选择这里的任意一个。

为了大家方便, webapp 模块默认包含了 Django 的模板系统. Django 模板是 Google App Engine 的一部分, 所以你不需要单独进行绑定就可以直接使用。

使用 Django 模板

首先在 `helloworld/helloworld.py` 中引入 `template` 模块

```
style="background-color: White;"  
  
import os  
  
from google.appengine.ext.webapp import template
```

重新编写 `MainPage` 处理过程:

```
style="background-color: White;"  
  
class MainPage(webapp.RequestHandler):  
    def get(self):  
        greetings = Greeting.all().order('-date')  
  
        if users.get_current_user():  
            url = users.create_logout_url(self.request.uri)  
            url_linktext = 'Logout'  
        else:  
            url = users.create_login_url(self.request.uri)  
            url_linktext = 'Login'  
  
        template_values = {  
            'greetings': greetings,  
            'url': url,  
            'url_linktext': url_linktext,  
        }  
  
        path = os.path.join(os.path.dirname(__file__), 'index.html')  
        self.response.out.write(template.render(path, template_values))
```

最后, 在 `helloworld` 的路径下创建一个新文件 `index.html`, 内容如下:

```

style="background-color: White;"
<html>
  <body>
    {% for greeting in greetings %}
      {% if greeting.author %}
        <b>{{ greeting.author.nickname }}</b> wrote:
      {% else %}
        An anonymous person wrote:
      {% endif %}
      <blockquote>{{ greeting.content|escape }}</blockquote>
    {% endfor %}

    <form action="/sign" method="post">
      <div><textarea name="content" rows="3" cols="60"></textarea></div>
      <div><input type="submit" value="Sign Guestbook"></div>
    </form>

    <a href="{{ url }}">{{ url_linktext }}</a>

  </body>
</html>

```

刷新，并查看结果。

`template.render(path, template_values)` 接受输入一个文件路径和一个dictionary类型的数据字典,并输出一段经过处理的文本。这个模板使用了Django模板的语法.在模板文件中可以直接使用引入的值，并且可以使用这些对象的属性。在许多实际例子里，你可以直接使用GAE数据模型，并访问他们的属性。

Tip: App Engine 应用程序对所有上传的文件的访问权限都是只读的，所以对文件的写操作是被禁止的；当前工作路径是应用程序的根目录，所以 `index.html` 的路径可以简单写成 `"index.html"`。

想要了解Django模板的详细语法，请查看 [the Django 0.96 template documentation](#)。

接下来...

绝大多数web应用程序都会输出通过模板动态生成的HTML代码。而大多数Web应用还需要处理静态文件，如：图像, CSS , JavaScript 脚本。为了了提高效率, App Engine 对静态文件

进行了特殊的处理.下一章, 我们会演示如果为这个应用添加CSS风格的支持.

Google App Engine 入门:使用静态文件

(本文译自:[Google App Engine Getting Started](#))

和其他的web发布环境不同, Google App Engine 不支持直接将应用目录下的文件直接输出的功能。也就是说, 如果我们将模板文件取名为 `index.html`, 我们并不能直接通过URL `/index.html`来访问这个文件。但是现在有非常多的应用需要我们提供文件直接输出的功能, 例如图片, CSS,JavaScript等等, 这些类型的文件都需要直接输出到客户端。GAE提供了这样的功能, 你不需要编写自己的处理模块来进行额外的处理。

使用静态文件

编辑`helloworld/app.yaml` 修改内容为:

```
application: helloworld
```

```
version: 1
```

```
runtime: python
```

```
api_version: 1
```

```
handlers:
```

```
- url: /stylesheets
```

```
  static_dir: stylesheets
```

```
- url: /*
```

```
  script: helloworld.py
```

新加的handlers 部分定义了两个URL处理模块, 其中 `/stylesheets`开头的URL都定义并转向了静态文件夹`stylesheets`, 如果在这个文件夹中发现了请求的文件, 就会直接把这个文件的内容返回给客户端; 而其他请求都会由`helloworld.py` 脚本进行处理.

默认情况下, App Engine 按照文件名后缀处理静态文件, 如 `.css` 结尾的文件就会使用MIME类型`text/css`.

GAE按照在 `app.yaml`定义的顺序对URL进行处理。在这个例子里 `/stylesheets` 将先于 `/*` 对路径进行处理。

想要了解更多在 `app.yaml`中的选项, 请查看 [the app.yaml reference](#).

下面, 我们创建 `helloworld/stylesheets`目录, 并且在这个目录下创建一个新文件 `main.css`:

```
body {
```

```
  font-family: Verdana, Helvetica, sans-serif;
```

```
background-color: #DDDDDD;
}
```

最后，编辑 `helloworld/index.html` 插入如下几行:

```
style="background-color: White;"
<head>
  <link type="text/css" rel="stylesheet" href="/stylesheets/main.css" />
</head>
```

刷新并查看效果

接下来...

我们的程序已经基本完备了，是时候把它发布到Google的服务器上去了。.

Google App Engine 入门:上传应用程序

创建和管理GAE应用程序，都需要使用GAE控制台程序来进行。其中，用于上传的命令行工具叫做 `appcfg.py`。

Note: 到目前为止，还没有办法删除已经发布的应用。这个功能会在稍后提供。在现在的预览测试阶段，每个用户可以注册3个应用程序ID,如果你不想使用你自己的应用程序ID,你可以仅仅读一下这一章，到你真正想发布自己的应用的时候再尝试。

注册应用程序

访问<http://appengine.google.com/>，使用你的Google帐号登录到App Engine管理平台。(如果你还没有Google帐号，请先[申请一个](#))

为了创建一个新的GAE应用，请点击按钮 "Create an Application" ，按照提示注册应用程序ID,应用程序ID的名字必须是唯一的。创建ID后，你就可以拥有一个<http://application-id.appspot.com/> 这样的URL地址来访问你的WEB应用了.当然，如果你拥有自己的域名的话，也可以将其绑定到你自己的应用。

修改 `app.yaml` ，吧 `application:` 的值设置为你刚刚申请的应用程序ID。

上传应用程序

执行命令程序:

```
appcfg.py update helloworld/
```

按照提示，输入您自己的Google 用户名和密码.

现在你已经可以使用如下地址访问您刚刚上传的WEB应用了

`http://application-id.appspot.com`

恭喜!

你已经完成了这个教程的学习了.想要了解关于Google App Engine的更多相关内容，您可以继续查看 [the App Engine documentation](#).