简介

tornado自带了一个简易的template引擎，使用它，我们可以很方便的构建自己的web页面。tornado的template使用起来很简单，如下:

```
from tornado.template import Template

t = Template("<html>{{ myvalue }}</html>")
t.generate(myvalue="Hello World")

#output: <html>Hello World</html>
```

可以看到，tornado的template很容易理解，我们加载一段模板代码，tornado会将该代码编译成python code。具体的过程如下：

- 解析template，生成code string
- 使用compile编译成code object
- 使用exec执行该code object

expression

上面的例子，我们可以看到，在template里面myvalue的值是在generate的时候传入并生成的。tornado通过{{ }}将python的值插入模板里面。也就是说，我们可以在双括号里面写上任何python表达式，tornado在模板生成的时候会计算表达式的值，并插入输出代码里面。如下：

```
Template("{{ 1 + 1 }}").generate()
# output: 2
Template("{{ Hello World }}").generate()
# output: hello world
Template("{{ [ i for i in range(10) ] }}").generate()
# output: '[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]'
```

control flow

不光支持表达式，tornado的template还可以通过{{% %}}来支持流程控制。tornado template支持的control有if，for，while和try，如下：

```
Template("{{% if a is 1 %}} Hello World {{% end %}}").generate(a = 1)
# output: ' Hello World '
Template("{{% if a is 1 %}} Hello World {{% end %}}").generate(a = 2)
# output: ' '

Template("{{% for name in names %}} {{ name }} {{% end %}}").generate(names = ['a', 'b',
 'c'])
# output: ' a  b  c '
```

export function

除了表达式和流程控制，tornado还给template提供了很多导出函数，可以在template里面直接使用，譬如最常使用的escape等，同时，我们还可以将函数传递给template，如下：

```
Template("{{ escape(url) }}").generate(url = 'a=1&b=2')
# output: 'a=1&amp;amp;b=2'

Template("{{ l(data) }}").generate(data = [1,2,3], l = lambda data: [ 2 * d for d in data ])
# output: '[2, 4, 6]'
```

inheritance

tornado的template支持继承，也就是说，我们可以通过继承一个基本的模板，然后再在子模板里面定制需要的东西。这个就跟c++里面多态一样。很是强大。

tornado的template通过extends和block来进行模板的继承。

```
#定义一个基本模板 main.html
<header>
{{% block header %}}{{% end %}}
</header>

#定义子模板 index.html
{{% extends main.html %}}
{{% block header %}} <h1>Hello world!</h1>
{{% end %}}
```

可以看到，template的继承很简单，我们在base模板里面定义整体的模板框架，使用block字段来标明子模板可以重载。子模板通过extends载入base模板，然后使用block来定制自己的功能。

UIModule

从上面可以看到，tornado的template虽然简单，但是却很强大，不光如此，tornado还提供了UIModule，让我们更强大的去定制自己的html页面。在编写不同的web页面的时候，有时候我们需要重用一些html代码，这种情况不适用extends和block，如果以c++对比，extends和block就如同类的继承，是类的复用，而我们这里则需要实现的是函数级别的复用。

幸运的是，tornado的UIModule提供了这种功能，我们首先继承UIModule，然后将其注册给application，这样模板就能使用这个UIModule了，如下：

```
class HelloModule(tornado.web.UIModule):
    def render(self):
        return '<h1>Hello, world!</h1>'

app = tornado.web.Application(
    ui_modules={'Hello', HelloModule}
)

#我们注册了一个Hello的UIModule，这样template就可以使用了,index.html
<html>
<body>
    {{% module Hello() %}}
</body>
</html>
```

可以看出，uimodule的使用也很简单，更强大的是，我们可以在uimodule里面嵌入自己的css，javascript等。只需要重载继承的embedded_javascript,embedded_css等，这里就不展开了。