**第20章Web编程**

Web编程分为Web客户端编程和Web服务器编程，Web客户端的功能大多由浏览器提供， 因此，本章着重对Web服务器编程进行讲解。

20。1 ft 么是 Web

Web服务器的实现离不开网络，与第18章网络编程中的服务器相同，Web服务器同样一 直等待客户端发送数据，当接收到客户端的请求报文后，Web服务器会提取报文中的数据信息, 对其进行处理，生成响应报文返回给客户端。那么，什么是Web服务器呢？

W3C (World Wide Web Consortium,万维网联盟)对Web服务器的定义为：Web服务器 是一个为了实现计算机之间的跨网络交互操作而设计的软件系统。简单地说，Web服务器是一 个位于物理服务器上的网络服务器。

20o2 Web

Web服务器程序与socket编程中基于TCP协议的服务器程序大致相同，不同的是，在返 回响应数据之前，需要先对数据进行封装。下面以一个可返回"hello itheima"的Web服务器为例， 展示Web服务器中数据的封装方式。

案例代码如下所示：

from socket import \*

#初始化主机地址与端口号

HOST, PORT = — 8000

def main():

server\_socket = socket(AF\_INET, SOCK\_STREAM)

server\_socket» setsockopt(SOL\_SOCKET, SO\_REUSEADDR, 1)

server\_socket»bind((HOSTf PORT))

server\_socket olisten(128)

print ( ,服务器%s启动 …。？ % PORT)

while True:

client\_sock7 client\_address = server\_socket.accept() #接收委戶端数据 — —

request = client\_sock.recv(1024)

#打印客户端数据包—

print(request.decode(\* gb2 312 ?))

#设置响应数据

http\_response = "n n

HTTP/1.1 200 OK

Helloz World!

#发送数据 "

client\_sock.send(http\_response.encode(J gb2 312 \*)) client\_sock o close()

if name == ! main ': main () .

以上案例中大部分代码与TCP服务器相同，特殊之处在于，服务器返回到客户端的信 息http\_response的格式是固定的，这条信息由"HTTP/L1 200 OK"、空行和字符串"Hello, World!"这三部分组成，若客户端成功接收到服务器返回的信息，信息中的字符串"Hello, World!"将会被打印到客户端。关于信息中每一部分代表的含义将在后续的章节中讲解，此处 读者有所了解即可。

执行程序,终端中打印的信息如下：

服务器8000启动

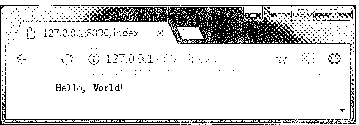
以上信息说明Web服务器已成功启动。在浏 览器的地址栏(此处以谷歌浏览器为例)中输入

图204浏览器页面

"127.0.0.l:8000/index” 并按［Enter ］键,浏览器跳 转的界面如图20』所示。

观察如图204,浏览器中成功打印岀"Hello, World!",由此可知Web服务器搭建成功。

20。咨 鏡一餐源定位简

图20」浏览器地址栏中的链接uhttp://127JJJ:8000/indexn又被称为统一资源定位符 (Uniform Resource Locator,简称 URL ) , URL 可分为三个部分,其中:

形“http”为协议头，指定本次请求使用的协议。

• "127.0.0.1:8000"为服务器地址，指定网络中唯一一台服务器。

-"/index"为资源路径，指定本次请求的资源在服务器中的位置。

URL中的协议头与客户端要实现的功能有关，当协议头为http时，表示本次请求的是基于 超文本传输协议的资源。超文本传输协议是Web编程中最常使用的协议，后面将对此协议进行 讲解。

20.4 HTTP«

为了保证服务器和客户端可以从接收到的消息中正确地解析数据，应使用约定的格式对数 据进行封装。根据客户端要实现的功能，应用层会使用不同的协议封装数据，其中最常用的协 议为超文本传输协议(HyperText Transfer Protocol, HTTP ) o

HTTP协议是一个应用层协议，它不传输数据，主要用于规定客户端和服务器发送数据的 格式。HTTP协议可以应用于使用TCP/IP协议簇的网络层之上(即应用层)，亦可用于其他 能保证可靠传输的网络中。在HTTP通信中，客户端可以是Web浏览器，也可以是其他基于 HTTP协议传输数据的软件。

服务器中存储着一些客户端可能会请求的资源。以浏览器作为客户端时，浏览器需要先与 Web服务器(HTTP协议的默认端口为80 )建立连接，之后Web服务器开始在指定端口监听客 户端请求。

浏览器发送请求给Web服务器之后，Web服务器会对接收到的消息进行处理，并生成响 应报文发送给浏览器，之后客户端再对响应报文中包含的资源进行解析，以网页的形式呈现到 浏览器中。

20o5 HTTP iiB

HTTP消息包括客户端向服务器发送的请求(request)消息，和服务器向客户端返回的响 应(response)消息。这两种消息都由一个起始行、一个或多个头域(HsdeQ、表示头域结束 的空行(HTTP协议中规定换行符为“\r\tT )和消息体(Body)组成。

消息中的每个头域由域名、冒号和域值三部分组成，下面是请求消息中的一个头域：

Host: 127.0^0^1:8000

这个头域表示客户端请求的服务器地址为127.001:8000。头域一般用于说明消息发送端的 属性信息，请求消息中的头域为客户端向服务器传递的关于请求或客户端的附加信息，响应消 •息中的头域为服务器向客户端传递的关于本次响应的附加信息。

在头域和消息体之间留有一空行，这个空行用于分隔头域和消息体，当客户端或服务器 从消息中读取到空行时，表示之后读到的将会是对方发送的消息体。可以在头域中使用域名 “Con知设置消息体的长度。请求消息的消息体可以为空，响应消息的消息体一般为 此次请求中客户端申请的资源。

起始行是HTTP消息中至关重要，亦是不可或缺的一部分。请求消息和响应消息的起始行 略有不同，下面将对这两种结构及其起始行分别进行讲解。

20.5J疇蕤g的

以下是一个HTTP请求消息:.

GET/first. jsp HTTP/1.1

Accept:image/gif.image/jpeg,\*/\*

Accept-Language: zh-CN

User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko

Accept-Encoding: gzip^ deflate

Host: 127.0 o 0.1:8000

DNT: 1

Connection: Keep-Alive

Cookie: —utma=96992031s914268212.1471315236.1497498941.1497516645.196; utmz=96992031.1471315236.1.1.utmcsr=(direct)|utmccn=(direct)|utmcmd=(none) username=j inqiao&password=1234

以上信息是由浏览器代为封装的请求消息，这段请求消息中的第一行为起始行。起始行指 明了本次客户端的请求方法(GET)、请求的资源(/firstjsp )及HTTP协议的版本(HTTP/L1 )。

起始行之后、空行之前的部分即为请求消息的头域，也称为请求头Request Headero空行 之后的部分为请求消息的消息体，也称为请求体(RequestBody)。

20.5.2矚廬鳍緻

以下是一个HTTP响应消息:

HTTP/1.1 200 0K '

Connection: Keep-Alive Content-Encoding: gzip

Content-Type: text/html;charset=utf-8

Date: Thu, 06 Jul 2017 06:59:54 GMT

Expires: Thu, 06 Jul 2017 06:59:54 GMT

Server: BWS/1o1

<html>

<head>

<title>itheima</title>

</head>

<body>

<hl style=ncolor:blue;°>hi</hl>

<p>hello itheima</pl>

<script type=ntext/javascript°> alert("hello")

</script>

</body>

</html>

以上信息中的第一行为响应消息的起始行，起始行给出了 HTTP协议的版本(HTTP/L1) 以及状态码(status cod© ),状态码由数字和字符串组成，数字表示请求处理结果(成功、失败 或其他)，字符串是对处理结果的说明。

起始行之后、空行之前的部分为响应消息的头域，也称为响应头(Response Head或)。空 行之后的部分为消息体,也称为响应体Response Body。

20S.3 HTTP .

HTTP 常用的请求方法有 6 种,包括 GET、POST、PUT、DELETE、OPTION 和 HEAD。

这些请求方法的含义分别如下:

。GET：向特定的资源发出请求。

* POST：向指定资源提交数据，待提交的数据被包含在请求体中，POST请求可能会在服 务器中创建新资源，或对已有资源进行修改。
* PUT：向服务器中指定位置上传资源。

。DELETE：请求在服务器中删除客户端URL标识的已有资源。

* PTION：请求返回服务器所支持的对特定资源的请求方法。

。HEAD。与GET类似，向特定资源发出请求，不同的是，使用此请求方法时不会返回响 应体。

需要说明的是，HTTP请求方法的名称区分大小写。以上方法中，除GET和POST方法外， 其他方法都是可选方法。

. 20D5a4

在访问20.2节中的服务器时，虽然URL中指定了资源路径“/index”，但事实上我们搭建 的服务器中并不存在这个资源，页面之所以能跳转成功，是因为服务器端设置了统一的响应信 息，只要客户端能成功与服务器建立连接，无论请求什么资源，服务器总是返回表示请求处理 成功的状态码“200 OK” ,返回统一的响应体"Hello, World!",因此，作为客户端的浏览器 则总会显示"Hello, World!" o

正式服务器会根据客户端请求的处理情况返回不同的状态码,状态码由3位数字组成，不 同的状态码代表不同的含义，具体分别如下：

-1 x x ：以1开头的状态码表示此次请求已经接受，需要继续处理，此类响应是临时响应。

° 2x x：以2开头的状态码表示请求已被服务器成功接收并处理。

* 3x x ：以3开头的状态码表示请求的资源在其他地方，需要客户端采取进一步操作才 能完成请求。
* 4x x ：以4开头的状态码表示请求出错。

° 5x x：以5开头的状态码表示服务器在处理请求的过程中产生错误或异常。

。600：表示服务器只返回了响应体。

常见的状态码有“200”和“404”，分别表示请求处理成功和未找到所请求的资源。

20。⑥ HTML »

什么是HTML?人们通常认为，网页就是HTML,但这种说法不够准确。HTML是 hyperText markup language的缩写,中文名称为“超文本标记语言”，简单地说,HTML是一 种搭建网页的语言。

网页中包含有丰富的资源，如文字、图片、视频等，为了在浏览器中呈现出这些网页资源, 且保证网页的排版布局美观友好，人们创建了 HTML。

HTML搭建的网页分为静态网页和动态网页，下面将从最基本的静态网页着手，简单介绍

HTML的语法规则。

20.6J静态飆甌

HTML语言之所以能够设置文本显示的格式，是因为其中预设了一系列代表不同格式的标 签。HTML中的标签由“。”和“</>”标识,如<html>和</html>,这样的标签通常成对出现。

下面是一段简单HTML文本：

<html>

<head>

</head>

<body>

<hl>hi</hl>

<p>hello itheima</pl>

</body>

</html>

上述文本共出现了 6种标签呀这些标签及其表示的含义分别如下：

•〈html〉标签：此标签与v/html>标签限定了文档的开始点和结束点，并告知浏览器当前 文档是一个HTML文档。

* vhead>标签：此标签与</hsd>标签用于定义文档的头部，它们是所有头部元素的容器。 •〈title〉标签：此标签与</title>标签用于定义文档的标题。

® <body>标签：此标签与</body>标签用于定义文档的主题，如文本、超链接、图像、表格、 列表等，这些内容又分别由不同的标签定义。

* <hl>标签:<hl>〜<16〉标签都可以定义标题,其中<hl>标签与</hl>标签用于定义 文档中的第一个标题，也是最大的标题，vh6>与</h6>定义最小的标题。

*翰*vp>标签:此标签与</p>标签用于定义段落,P元素(vp>标签定义的内容)会自动在 其前后创建一些空白。

除以上这6种标签外，HTML中还定义了许多具有丰富含义的标签，有兴趣的读者可查阅 资料自行学习。

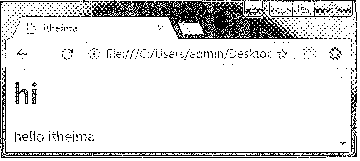
将以上的HTML文本保存在文件test.html中, 在浏览器中打开该文件，显示的网页如图20.2所示。

图 20-2 test.html

在图2()『2所示的页面中，文档标题为“ithdma”, 一级标题为“hi”，文本中的段落为"hello itheima" ° 类似test.html这种只能展示文件中已经编辑好的信息 的网页被称为静态网页。

20b®d2 css a#

CSS是层叠样式表(Cascading Style Sheets )的简称,是一种表现HTML、XML等文件样 式的计算机语言。

在2061小节中实现的静态网页只能显示简单的黑色文本，但我们日常生活中接触到的网

页往往是丰富多彩的，网页中的文字可能有不同的字体、颜色和尺寸，若想创建这样的网页，

CSS必不可少。

以20.6J小节创建的网页为例，使用CSS将vhl>标签中的文本设置为24号字并添加阴影，

示例如下:

<html>

<head>

<title>itheima</title>

<style>

hl{

font-size: 24px; text-shadow: 2px 2px }

</style>

</head>

<body>

<hl>hi</hl>

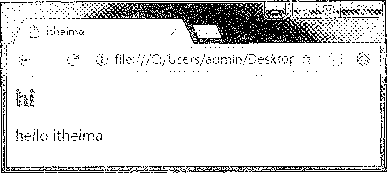
<p>hello itheima</pl>

</body>

</html>

以上示例在原文本的基础上添加了 <style> 标签,<style>标签用于为HTML文档定义样式 信息，该标签中重新定义了 <hl>标签的样式。 在浏览器中打开testlitml文件，浏览器中展示的 网页如图20・3所示。

比较图20-2和图20-3,图20-3中一级标题 “hi”的字体发生了改变，且新增了阴影，可知

<hl>标签的样式设置成功。

2px #666666;

图 20-3 test.html

20D®n3 JawaSoiptW#

在某些情况下，人们不只希望能编写美观友好的网页，还希望网页能与用户交互，根据用 户的动作产生不同的网页。此时，仅使用HTML和CSS是不够的，还要用到脚本。JavaScript 脚本是HTML中最常用的一种脚本，简称JSO

JS脚本即可以内嵌在HTML文本中,也可以从外部链接到HTML中。下面以2062小节 的test^tml文件为基础，在其中添加JS脚本语言，实现用户单击网页标题时，标题变为蓝色的 效果。示例如下：

<html>

<head>

<title>itheima</title>

<style>

hl{

color: #222222; font-size: 24px; text-shadow: 2px 2px 2px #666666;

</style>

<script> function change() {

document. getElementsByTagName ( v hl? ) [ 0 ] .. style . color = 1 #0000f f \* ; }

</script>

</head>

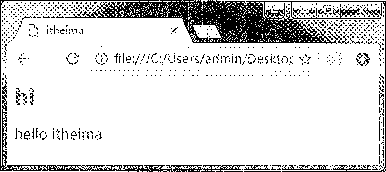
<body>

<hl onclick="change()">hi</hl>

<p>hello itheima</pl>

</body>

</html>

保存文件，并在浏览器中打开修改后的文件， 当前的网页与20.6.2小节中图20-3显示的网页 相同。单击网页中的标题“hi”，此时显示的网 页如图2CM所示。

观察图2CW■，网页中的标题“hi”显示为蓝色， 可知程序实现成功。

本节讲述的HTML、CSS以及JavaScript都

是与Web编程中前端搭建的相关知识，虽然Web服务器侧重后台，但这些前端知识也是读者

必须了解的。关于前端的部分就讲解到此，读者若有兴趣，可自行查阅资料学习。

20o 7 » Web

Web服务器可分为静态Web服务器和动态Web服务器，当然，一个完善的Web服务器应 既能处理静态资源，又能处理动态资源，但为了降低读者的学习难度，本节将从最基础的静态 Web服务器入手，对Web服务器进行讲解。

静态Web服务器只能返回服务器端存在的一些静态资源，如一段文字、一些图片或静态 网页。在202小节中实现的Web服务器，就是一个静态Web服务器。

202小节的静态服务器中响应信息固定不变，但实际开发中，客户端请求的数据一般存储 在服务器端的文件中，因此服务器应根据客户端的请求读取相关资源文件，再将读取到的资源 与状态码和头域等组成响应消息，一起发送给客户端。

下面通过一个实际案例来展示基于文件的静态Web服务器的搭建方法。其中服务器代码 存储于文件static\_web\_server\_file.py，待返回的网页信息存储于文件hello.html中。

(1 ) Web 服务器文件 static\_web\_server\_file.py

服务器程序的代码具体如下:

1. from socket import \*
2. from multiprocessing import Process
3. import re
4. import sys
5. #设置静态文件根目录

HTML\_ROOT\_DIR = "./" #初払化主風地址与端口号

#将当前目录设置为根目录

HOST, PORT =七 8008

def handle\_client(client\_socket):

"° °处理客户端请求° ° " -

#获取客户端请求数据

request\_data = client\_socket» recv(1024) request\_\_lines = request\_data . splitlines ()

# \*GET / HTTP/1.1,

#解析请求报文

request\_line = request\_lines[0]

#提取用另请求的文件名 —

file\_name = re .match (r ' \w+\s+ (/ [A ) \ s \* z

request\_line .. decode ( 5 gb2 312 \* ) ) . group (1) file\_name = HTML\_ROOT\_D IR + file\_n.ame

：駐 i f条件语句中少写等号时发生赋循

if , / ? == 'file\_name !:

file\_name = 1 / index »html1

#打开文件，W卖取内容

try:

# file = open (HTML\_ROOT\_DIR + file\_name, ? rb , ) print (file\_name) file = open (file\_name^ ! rb ,)

except lOError : #如果出现异常

#构造响应数据

resp\_start\_line = "HTTP/1o1 404 Not Found\r\n" resp\_handers = "Server:My server\r\n" resp\_body ="未找到指定文件!"

else:

file\_data = file » read ()

file . close ()

#构造响应数据 resp\_start\_line = "HTTP/l.1 200 OK\r\n° resp\_handers = "Server:My server\r\n" resp\_body = file\_data. decode (f gb2312 1)

resp = resp\_start\_line + resp\_handers + ?\r\n, + resp\_body print("%s" % resp)

#向客户端返回响应数据

client\_socket ” send(resp.encode(J gb2312!))

#关闭客户端连接

client\_socket。close()

def main():

server\_socket = socket(AF\_INETZ SOCK\_STREAM) server\_socket.bind((HOST, PORT))

server\_\_socket.setsockopt(SOL\_SOCKETZ SO\_REUSEADDRZ 1) server\_socketlisten (128) print ( !服务器8008启动.L)

while True:

client\_socket f client\_address = server\_socket.accept() print (n 有用户连接到达：%s" % str (client\_address)) handle\_\_client\_process = Process(target=handle\_clientz args=(client\_socketz)) handle\_client\_process . start () # 开启进程

client\_socket.close()

1. if name == " main n :
2. main ()

在python中，Web服务器的根目录可通过系统变量HTML\_ROOT\_DIR进行设置，服务器 文件第6行代码中将当前目录设置为服务器的根目录。

代码分析如下：

服务器文件中包含两个函数，其中maiiiO函数用于实现socket通信的基本流程，handle\_ clientQ函数用于处理客户端的请求。

在handle\_client()函数中，第12行代码用于获取客户端的请求消息，第13行代码将请求 消息按行拆分,并存储于列表request\_lines中。第16行代码从列表request lines中获取请求信 息的起始行，第18行代码从起始行中提取资源所在路径，此时获取的路径是以HTML\_ROOT\_ DIR为根目录的相对路径，服务器查找文件时会以绝对路径进行查找，因此在第20行代码中 将其与HTML\_ROOT\_DIR进行拼接。

获取文件所在路径后，可使用文件操作函数打开文件，读取文件内容。因为请求的资源并 不一定存在，所以第25〜40行代码在操作文件的同时设置了容错处理，若文件打开失败，状 态码将被设置为“404”，响应体被设置为“未找到指定文件！ 若文件打开成功，则读取

文件内容，将状态码设置为“200”，并将响应体设置为文件内容。

第41行代码对响应数据进行了拼接，第42行代码将响应信息输出，第44行代码将响应 信息发送到客户端，第46行代码关闭客户端socket,断开连接。

(2 )静态网页文件 index.html

静态网页文件index.html中的内容如下:

1. <!DOCTYPE html>
2. <html lang="enn>
3. <head>
4. <meta charset=°UTF-8n>
5. <title>My Web</title>
6. </head>
7. <body>
8. <hl>hi</hl>
9. <p>welcome</p>
10. </body>
11. </html>

将index^html文件与服务器文件放在相同路径下，启动服务器，在网页地址栏中输入 “ 127.0.0.1:8008/index.html"并按［Enter ］键,浏览器中显示的网页如图20-5所示。

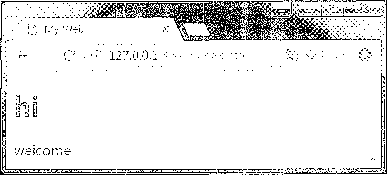
由图2()・5可知，浏览器成功向服务器发送 请求，并获取到服务器返回的消息。

图20-5服务器返回到客户端的index.html

查看启动服务器的终端，终端中打印的信息 如下：

服务器8008启动•…

有用户连接到达:(\*127.0.0.34209)

.//index。html

HTTP/l.1 200 OK

Server:My *server*

<!DOCTYPE html>

<html lang=nen°>

<head>

<meta charset=°UTF-8"> <title>My Web</title>

</head>

<body>

<hl>hi</hl> <p>welcome</p>

</body>

</html>

终端打印的信息中包含服务器的启动信息、用户连接信息和服务器响应信息。观察其中的 响应信息：响应码为"200 OK",可知本次请求处理成功；响应体为文件index.htmi中的内容， 可知服务器成功访问并获取了文件index„htmlo

若客户端访问的文件不存在，如在浏览器地址栏中输入“ 12700.1:8008/index”，则服务 器将会返回第31〜33行代码设置的响应消息，此时终端中打印的信息如下：

有用户连接到达:(?127.0.0.1\ 34510)

.//index

respone:HTTP/l.1 404 Not Found

Server:My server

20o8 W跡］I规龜

经过以上的学习，读者应已了解了 Web编程的大致流程：

(1 )客户端发送一个HTTP请求；

(2 )服务器接收并处理客户端请求，生成一个HTML文档；

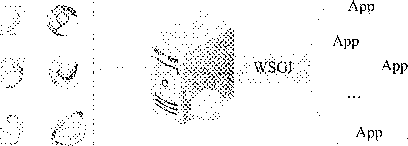
1. 服务器将HTML文档作为HTTP响应的响应体发送给客户端；
2. 客户端接收到HTTP响应，获取其中的HTML文档，根据HTML文档在客户端中生 成页面。

分析以上流程可知，最简单的Web服务器中只需先存储一些客户端可能会请求的HTML文 件，再实现接收、处理客户端请求，以及返回响应消息的功能即可。但这种服务器只是一个静态 服务器，无法返回动态资源,若客户端请求的资源需要服务器动态生成，就需要实现动态服务器。

事实上，无论是静态服务器还是动态服务器，都需要实现接收请求、处理请求、返回响应 信息的功能，它们的主要差别在于要返回的HTML文件是否需要动态生成。因此，在实际Web 开发中，接收请求、处理请求和返回响应信息这些底层功能通常由服务器软件实现，而开发人 员则专注于开发能动态生成HTML文档Web应用。

Pythen Web服务器能处理的请求应是极其丰富的9 Web应用的开发可能由多名开发人员实 现。为了保证开发人员能将重心放在HTML文档之上，且每个开发人员编写的Web应用都能以简单的方式被服务器调用，需要为所有的Web应用统一接口。

Web编程中常用的Web应用统一接口为WSGI (服务器网关接口 , Web Server Gateway In­terface ),实际上，WSGI是一种规范，它规定了 Web应用接口的格式，只要开发人员在Web 应用中实现一个符合WSGI规范的函数，这个应用就可以在WSGI服务器中使用。

客户端、服务器、WSGI以及Web应用之间的关系如图20・6所示。

客户端 Web服务端

Web应用

客户端、月艮务器、WSGI以及Web应用的关系图示

图 20-6

WSGI规定的Web应用接口的格式如下:

def application(envzstart\_response):

status = "200 OK"

headers =[

("content-Type",°text/plain")z

]

start\_response(status,headers) return H<hl>hello itheima</hl>°

以上定义的application()函数是一个符合WSGI的请求处理函数，这个函数中接收两个参数 *會*env： 一个包含HTTP请求信息的字典对象。

® start response: 一个由服务器传递给Web应用,用于从Web应用中获取响应消息头域 信息的函数。

start\_response()由服务器传递给Web应用,并将从Web应用中获取的状态码和头域信息反 馈给服务器，因此start\_response()函数应是由服务器定义的回调函数，该函数中包含两个参数： status和headers。其中status是一个表示状态信息的字符串，headers是表示头域信息的列表。此夕卜, 服务器返回的响应消息只有一条，因此在Web应用中start\_response()函数只能被调用一次。

以上示例中applicationQ函数的返回值是一行HTML文本信息，这条信息将被作为HTTP 响应的响应体发送给浏览器。

符合WSGI规范的Web应用程序中,函数application()通常由WSGI服务器调用,在下面 的小节中，我们将对WSGI服务器进行讲解。

I 20。联 WSGI

由于WSGI服务器中所有的Web应用都符合WSGI规范，所以服务器只需根据获取的 URL中提供的资源信息，便可调用不同Web应用中的接口。下面通过案例来展示动态Web服

务器的搭建方法,此案例包含一个服务器文件dynarnics\_web\_server\_file.py和两个Web应用文 件c\_time.py和say\_hello.py?应用文件的功能分别如下:

• c time.py:获取系统当前时间。

® say hello.py： 向客户端发送 "hello itheima”。

(1 )月艮务器文件 dynamics\_web\_server\_file.py

动态Web服务器文件中的代码如下所示。

from socket import \*

from multiprocessing import import re

import sys

HTML\_ROOT\_DIR = "

PORT = 8000 class HTTPServer(object): def

Process

#将当前目录设置为根目录

def

def

def

init (self):

° " ° 初女台彳匕° ° °

self.server socket

=socket(AF\_INET, SOCK\_STREAM)

self o server\_socket o setsockopt (SOL\_SOCKETZ SO\_\_REUSEADDR,F 1) bind(selfz port):

s顷绑定服务器地址心顷

self . server\_\_socket.. bind ( ( n F/ 7 port))

start(self):

"° °启动服务器尹处理请求° n "

self o server\_socket.listen(128) print (n月艮务器％s开启 叫% PORT)

while True:

client\_socketz client\_\_address = self «server\_socket.accept() print ("有客户端连接到达:%s° % str (client\_address)) handle\_client\_process = Process(target=self.handle\_\_clientz args=(client\_socketz)) handle\_client\_proeess.start()

client\_socket.close() handle\_client(selfz client\_socket):

""n处理已连接客户端的请求f, ° °

#获取客户端请求数据

request\_data = client\_socket.recv(1024) request\_lines = request\_data o splitlines() request\_start\_line = request\_lines[0]

#提取用户请求時文件名 ~

file\_\_name = re .match (r ! \w+\s+ (/ [A ]\*)\s?’

request\_start\_line•decode(\* gb2 312!)) .group(1) if file\_name . endswith (" »pyn ):

# Wf py文件

m = import (file\_name [ 1: - 3 ])

#定司境变量~T字典美型)

env = {} response\_body = m. application(env^ self.start\_response) response = self.response\_headers + n\r\n" + response\_body

#向客户端返回响应数据 - —

client\_socket。send(response。encode(5 gb2312'))

#关闭客户端连接 client\_socket.close() def start\_response(selfstatus, headers):

心顷从W㊀b应用中获取响应码及响应体头域e*顷* response\_\_headers = "HTTP/1.1 " + status + 1 \r\n \* for hreader in headers: response\_headers += "%s: %s\r\nn % hreader self. response\_headers = response\_header,s def main():

http\_server = HTTPServer()

http\_server.bind(PORT) http\_server.start() if name == " main n : main ()

以上代码中将服务器被抽象成了一个HTTPServer类，这个类中定义了五个方法，它们的 功能分别如下：

* —init : init 方法用于初始化类对象，在类对象被创建时，程序中会调用socketQ 函数创建服务器socket,并设置套接字可重复使用。

® bind：该方法重写了 socket模块中bind()函数，它只需接收一个端口号port,就能为服 务器socket绑定地址。 '

* start： start方法用于启动服务器，处理客户端连接请求，并创建子进程处理已建立连接 的客户端的资源请求。

*Q* handle\_client：它是加rt方法中创建的子进程的功能函数，此方法接收一个用于与客户 端交互的套接字clienVock作为参数。hand虹client方法的功能是接收客户端发送的请 求消息，从中提取脚本文件名°将该脚本作为模块导入服务器，由服务器调用模块中的 应用函数生成动态响应消息，并发送到客户端。

* start\_response：它是一个回调函数，当服务器调用Web应用中的函数时，该方法作为函 数的参数被传递到Web应用中9获取Web应用中的状态码和头域信息。

服务器代码中除了定义HTTPServer类外，还定义了 main。函数，该函数中创建了 HTTP- 类的实例，调用bind方法为服务器绑定地址，并调用start方法启动了服务器。

(2 ) Web 应用文件 c\_time.py

用于获取系统当前时间的Web应用文件中的代码如下所示。

import time

def application(env^ start\_response): status = "200 OK"

headers =[

("Content-Type"r "text/plain")

start\_response(status zheaders) return time.ctime()

(3 ) Web 应用文件—say hello.py

向客户端发送“hello itheima"的Web应用文件中的代码如下所示。

452 Pyfthoo雲曬疆隣 關饕寥Pjth©o

def application(envzstart\_respouse):

status = "200 OK"

headers =[

("Content-Type"f"text/plain")

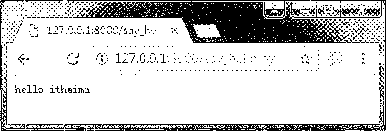
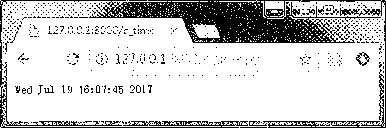
] start\_response(status fheaders) return "hello itheima"

将外部程序文件与服务器文件置于同一目录下，启动服务器，在网页中输入u 127.0.04:8000/

c\_tim&py"，服务器中打印的信息如下： 服务器8000开启

有客户端连接到达：(,127.0.0.1,, 27247)

有客户端连接到达：(,127.0.0.1\ 27248)



浏览器中展示的网页如图20"所示。

在网页中输入 “127.0.0.1:8000/say\_hello.py,,

图20-7动态获取系统时间

,浏览器中展示的网页如图20.8所示。

图 20-8 动态返回"hello itheima”

由以上程序执行结果可知，当用户在客户端请求不同资源时，无须修改服务器代码，服务 器便能返回正确的响应结果，说明WSGI服务器实现成功。

20oI0 Web IOB

一个成熟的Web服务器应当既能返回静态网页，亦可返回动态数据。若想使服务器可具 备这两种能力，需要对从请求消息中提取到的资源文件名进行分析须获取文件类型，再根据获 取的文件类型，执行不同的流程。

下面将通过一个案例来展示搭建能同时处理静态数据请求和动态数据请求的Web服务器 的方法*，*该案例存储于文件web\_serveLpy中,具体实现如下:

1. from socket import \*
2. from multiprocessing import Process
3. import re
4. import sys
5. HTML\_ROOT\_DIR = L/，， #将当前目录设置为根目录
6. PORT = 8000
7. class HTTPServer(object):
8. def init (self):
9. " ° "初始化H " n
10. self.server\_socket = socket(AF\_INET7 SOCK\_STREAM)
11. self.server\_socket.setsockopt(SOL\_SOCKET, SO\_REUSEADDR, 1)
12. def bind(selfF port):
13. 5 °绑定服务器地址° "

self.server\_socket.bind((""f port))

def start(self):

心顷启动服务器«处理请求° ° °

self.server\_socket.listen(128) print ("服务器 %s 开启..."% PORT) while True:

client\_socketz client\_address = self.server\_socket.accept() print (° 有客户端连接到达:%s" % str (client\_address)) handle\_client\_process = Process(target=self.handle\_client7 args=(client\_socketz)) handle\_client\_process.start()

client\_socket.close()

def handle\_client(self, client\_socket):

""n处理已连接客户端的请求"""

#获取客户端请求数据

request\_data = client\_socket.recv(1024) request\_lines = request\_data.splitlines() request\_start\_line = request\_lines[0]

#提取用乒请求"文件名 —

file\_name = re .match (r \* \w+\s+ (/ [A ]大)\s '

request\_start\_line.decode(1gb2312*，)).*group(1) if file\_name . endswith (° . py" ) : # 请求获取脚本资源

# Wfpy文件

m = import (file\_name [ 1: -3 ])

#定又毓境变量字典美型)

env = {} response\_body = m.application(envz self.start\_response) response = self.response\_headers + "\r\n" + response\_body else :

if , / 1 == file\_name : file\_name = 1 /index. html'

#打开文律z读取内容

try:

file = open (HTML\_ROOT\_DIR + file\_name, ' rb 1 ) except lOError : #如果出现异常

#构造响应数据 responce\_start\_line = "HTTP/l„1 404 Not Found\r\n" respheaders = "Server:My server\r\n" responce\_body = °未找到指定文件！"

else :

file\_data = file . read ()

file . close ()

#构造响应数据 responce\_start\_line = "HTTP/1«1 200 0K\r\n" responce\_headers = "Server:My server\r\n" responce\_body = file\_data . decode (\* gb2312 1)

response = responce\_start\_line + responce\_\_headers +

\*\r\n \* + responce\_body

#向客户端返回响应数据 client\_socket.send(response.encode(\* gb2 312'))

1. #关闭客户端连接
2. client\_socket.close()
3. def start\_response(selff statusz headers):
4. "5 从Web应用中获取响应码及响应体头域° ""
5. response\_headers = "HTTP/l.1 " + status + 1\r\n
6. for hreader in headers:
7. response\_\_headers += "%s: %s\r\n" % hreader
8. self . response\_\_headers = response\_headers
9. def main():

http\_server = HTTPServer() http\_server.bind(PORT) http\_server.start()

73

74

75

7 6 if name == " main

77 main()

启动服务器，在浏览器地址栏中输入"127.0JJ:8000/c\_time.pyn 9浏览器展示的网页如

图20・9所示。

在浏览器地址栏中输入u127.0„0.1:8000/indexhtmr5 9浏览器展示的网页如图20』。所示。

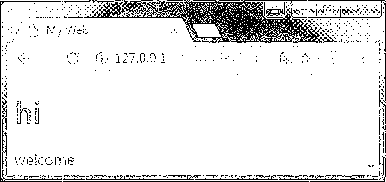
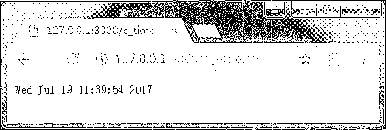


图20項获取动态数据

图2CM0获取静态数据

结合图20・9和图2040中的网页可知，既能返回静态数据，又能返回动态数据的Web服 务器实现成功。

20ol 1基于框禁的®箋器

什么是框架？在建筑学概念中，框架(FTamewoik)是一个起到约束作用的框子，一个具 有支撑线的架子。其实，框架的概念在软件工程中也被广泛应用，在软件工程中，框架通常被 认为是已经实现某应用领域通用功能的底层服务，在此基础之上，开发人员可以按照某种规则 对软件进行扩充,以达到缩短开发周期、提高开发质量的目的。

开发人员若想在20J0小节实现的Web服务器的基础上添加新的功能，只需创建新的Web 应用程序，在其中定义遵循WSGI规范的、名为application的函数，而无须更改服务器程序， 可以说，这个服务器已经使用了框架的思想，实现了程序的复用。但这个服务器的实现方式与 日常生活中接触到的浏览器相比仍有不足：

1. 读者在浏览网页时可能会发现，浏览器地址栏中出现的一般不是具体的、带有后缀 的文件名，而是一个字符串，也就是说，我们日常生活中接触到的Web服务器实际上通过一个 字符串而非文件名来判断客户端要请求的资源。
2. 服务器需要使用的请求处理函数存储在不同的文件中，因为函数名全部相同，应用 文件无法合并，且无法反映应用的实际功能。
3. 服务器不够精简，除接收和发送数据外，还要实现请求分发的功能，对服务器的效 率造成了 一定影响。

那么，如何才能实现一个效率更高、与常用浏览器功能更为相似的Wd＞框架呢？

20JU W®b

Web服务器和客户端的功能其实都是固定的：客户端用于发送请求、接收响应消息；服务 器用于接收请求，发送响应消息。因此，Web框架其实是Web应用的抽象，框架从服务器获取 客户端发送的请求，并根据请求中携带的信息，匹配请求处理函数，对请求进行处理。

Web应用框架的雏形如下所示。

def application(envr start\_response):

urls =[

(n/ctime", ctime),

(n/sayhello"f sayhello)

path = env.get(°PATH\_INFO")

for urlf app in urls:

if path == url:

app 0

以上框架中设置了一个名为uHs的列表，列表中的元素为元组，元组中包含两项内容9分 别表示URL中出现的资源路径和该脚本对应的功能函数，这个uHs类似路由器；urls列表之后 使用变量path从参数env中获取本次URL中附加的资源路径信息，并在循环中遍历wds列表， 获取到该列表中的每一个元素，将元素中管道资源路径与本次请求中URL的资源路径信息进 行匹配，如果匹配成功，执行列表元素中的对应的Web应用函数。

需要说明的是，在符合WSGI规范的情况下，Application必须是一个可调用对象，因此它 不仅可以是一个函数，也可以是一个类。下面结合代码来学习如何将Web应用文件抽象为一个 框架，代码如下所示。

1. import time
2. HTML\_ROOT\_DIR = "./°
3. clsiss Application (object):

| 4 | "5框架主体程序nun |
| --- | --- |
| 5 | def init (selfzurls): |
| 6 | #设置路由信息 |
| 7 | self.urls = urls |
| 8 | def \_call (selfenvf start\_response): |
| 9 | path = env.get (nPATH\_INFOn, n/n) # 设置默认值为根目录 |
| 10 | #映射 \_ |
| 11 | for urlr handler in self.urls: |
| 12 | #(n/ctime", show\_ctime) |
| 13 | if path == url: |
| 14 | return handler(env7 start\_response) |
| 15 | #未找到路由信息，404错误 |

#』夺当前目录设置为才艮目录

1. status = "404 Not Found"
2. headers =[]
3. start\_response(statuse headers)
4. return "not found"
5. #Web app—show\_ctime()
6. def show\_ctime(env7 start\_response):
7. status = "200 0K°
8. headers =[
9. ("Content-Type"f °text/plain")
10. ]
11. start\_response(status,headers)
12. return time.ctime()
13. #Web app—say\_hello()
14. def say\_hello (envz start\_response.):
15. status = "200 OK"
16. headers =[
17. ("Content-Type","text/plain°)
18. ]
19. start\_response(statusfheaders)
20. return "hello itheima "

3 6 #设置路由列表

1. urls =[ .
2. (°/,\ show\_ctime)f
3. (H/ctimenz show\_ctime),
4. ("/sayhello",say hello)
5. ] —
6. #生成Web应用对象
7. app = Application(urls)

以上代码存储于文件WebFramework.py中,主要包含一个Application类、两个Web应用 函数 show\_ctime()与 say\_hello()和路由列表 urls。

Application类中实现了两个方法,其功能分别如下:

* —init : init 方法用于初始化Application类的对象，该方法接收一个参数urls, urls是一个路由列表。
* 方法使Application类的对象在程序中可以像函数一样被调用，注意 到这个方法的参数列表与WSG][规范中定义的接口的参数列表相同，实际上，\_call\_ 方法相当于所有Web应用函数的统一接口。在\_罗疝\_\_方法中，程序从参数env中获取 URL中请求的资源，在循环中寻找与资源对应的功能函数，返回请求处理结果。

show\_ctime()与say\_hello()是Web应用函数,show\_ctime()可以获取系统当前时间,say\_ 損恥()可以生成动态响应体"hello itheima"。路由列表iris中存储了浏览器中可能出现的字符 串与应用函数之间的映射关系。

20d11q2 .

框架只是服务器的支撑，它并不能替代服务器，因此，在实现了框架后，还需要编写一段 服务器程序，并在启动服务器时调用框架。

服务器的代码实现如下所示：

from socket import \*

from multiprocessing import Process import re

import sys

HTML\_ROOT\_DIR =*、'/* #将当前目录设置为根目录

PORT = 8000

class HTTPServer(object):

|  |  |
| --- | --- |
| def | init (selfr application):  ° ° "构造函数,application指的是框架中的类application" ° ° self.server\_socket = socket(AF\_INET/ SOCK\_STREAM) self.server\_socket.setsockopt(SOL\_SOCKETZ SO\_REUSEADDR, 1) self.app = application |
| def | bind(selfz port):  self.server\_socket.bind(("", port)) |
| def | start(self):  self.server\_socket.listen(128)  while True:  client\_socket, client\_address = self»server\_socket.accept() print (” 有客户端连接到达:%s° % str (client\_address)) handle\_client\_process= Process(target=self.handle\_clientz args=(client\_socket,))  handle\_client\_process.start()  client\_socket.close() |
| def | handle\_client(selfz client\_socket):  "° °处匝客户端请求° ° "  #获取客户端请求数据  request\_data = client\_socket.recv(1024) request\_lines = request\_data.splitlines()  # 'GET / HTTP/1.1\*  #解析请求报文  request\_start\_line = request\_lines[0]  #提取用乒请求"文件名 ~  file\_name = re .match (r1 \w+\s+ (/ [A  request\_start\_line.decode(!gb2312')).group(1) #定义环境变量(字典类型)- -  env = {  °PATH\_INFOn :file\_\_name,  "METHOD" : "method.0  } response\_body = self.app(env, self.start\_response) response = self.response\_headers + °\r\n" + response\_body #向客户端返回响应数据  client\_socket.send(response .encode(1gb2312 \*))  #关闭豪户端连接  client\_socket.close() |
| def | start\_response (self；, status, headers): response\_headers = "HTTP/1.1 " + status + \*\r\n1 for hreader in headers:  response\_headers += "%s: %s\r\n"%hreader self.response headers = response headers |

1. def main():
2. if len(sys.argv) < 2:
3. sys.exit("python Webserver.py Module:app")

• 54 # python Webserver.py WebFrameWork:app

55 module\_namef app\_name = sys.argv[1] split (":")

1. 6 # module\_name = "WebFremeWork"
2. # app\_name = °app"
3. m = import—(module\_name)
4. #获取app对象
5. app = getattr(mz app\_name)
6. http\_server = HTTPServer(app)
7. http\_server ..bind (PORT)
8. http\_server.start()
9. if name == " main ":
10. main()

以上代码存储于文件WebServer.py中,其中实现的服务器与20.10小节实现的服务器大同 小异,不同的是,WebServer.py中的HTTPServer类在实例化时需要传入一个框架中Application 类的实例。

20J1.3瀛褸翁毓

服务器程序从主函数mahi()开始执行，主函数需要接收两个参数，第一个参数为可执行 的服务器文件.第二个参数为“框架名：app”，其中框架名是Web应用框架的文件名，app为 Web应用框架中创建的Application类的实例。

上个小节中第52行代码对main()函数接收到的参数的数量进行判断，若少于2,则退岀 进程，提示用户服务器启动命疏令的正确格式。若接收到的参数数量不小于2,主函数开始执行 第58行代码。

第55行代码分别通过参数module name和app name接收Web应用框架的文件名 (WebFramework)以及服务器框架中创建的Application类的实例(app)。第58行代码中, 使用\_import\_()函数导入框架模块；第60行代码中，通过getatt『()函数获取框架模块中创建 的对象并赋给变量app；第61行代码实例化HTTPServer类对象，创建服务器，并使用从框架 中获取的Application类对象app初始化HTTPServer类对象；第62行代码为服务器绑定端口 ; 第63行代码启动服务器。

服务器启动后会执行一系列TCP服务器通用流程一一接收客户端连接请求、为客户端创 建子进程、在子进程中处理客户端请求、通信完毕后断开连接，关闭客户端。

通信流程中对客户端请求的处理由HTTPServer类中的handle\_client方法实现,在handle\_ client方法中，服务器首先接收客户端发送的数据，之后从其中获取用户请求的资源存储在变 量fileName中。在第40行代码中，服务器调用了 Web应用框架中传递过来的类对象app9该 对象被调用时执行\_\_喝11\_方法处理了客户端请求，并通过回调函数start\_response()为服务器 设置了本次请求处理后待发送的响应消息中的状态码和头域，通过返回值将响应体返回到服务 器。第41行代码服务器程序将响应头和响应体进行拼接，第43行代码服务器将响应消息发送

到了客户端，第45行代码关闭客户端连接。

至此，服务器对客户端的一次请求处理全部完成。

20J1.4购離麗素

在项目所在路径打开终端，输入如下所示的启动命令:

python Webserver.py WebFramework:app

在浏览器地址栏中输入“127。0」:8000”，浏览器中生成的界面如图2(M1所示。 终端中打印出如下所示的连接信息:

有客户端连接到达：(U27.0.0.1S 5010)

有客户端连接到达：(?127.0.0.1\ 5014)

继续测试Web框架中的其他功能。在浏览器地址栏中输入“127.0.0」:8000/ctime”，浏览 器中生成的界面如图2042所示。

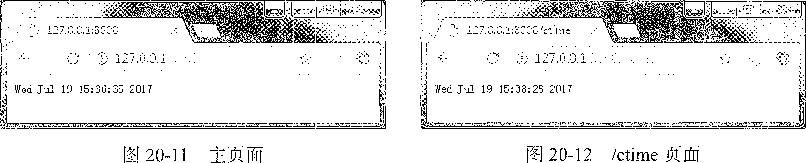


图20-11与图20-12中所示的界面基本完全相同，这是因为,在框架程序的路由列表urls中,

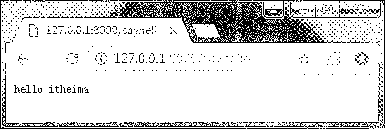


图 20-13 /sayhello 页面

将 *T* 和“/ctime”的请求都映射给了 show\_ctime()函数。

在浏览器地址栏中输入a127JJ.l:8000/sayhellon，服务器会根据路由列表iris调用框架 中实现的say\_liello()函数,此时浏览器中生成的 界面如图2()』3所示。

本节中实现的Web服务器是一个动态Web 服务器，只能生成动态页面，若想使其同时能够 返回静态页面，可以在iiHs中添加从字符串到静 态文件名的映射，之后在方法中对匹配 到的handler的类型进行判断，若handler是一个函数，则直接在return语句中调用函数；若 handler是一个字符串须则打开与字符串同名的文件，读取并返回文件中的内容，此处不再给岀 代码实现。