3.1 网站树的爬取路径

一个网站往往由很多相互关联的网页组成，每个网页上都可能包含我们锁要关心的数据，那么我们怎么样获取这些数据呢？显然我们必须穿梭于各个网页之间，那么按什么样的规则穿梭呢？常用的有深度优先与广度优先方法。为了说明这两种方法的工作过程，我们特意设计一个简单的网站。

3.1.1 Web 服务器网站

我们设计好 books.htm, program.htm, database.htm, netwwork.htm, mysql.htm, java.htm, python.htm 等网页文件以 utf-8 的编码存储在文件夹（例如 c:\web)中，各个文件的内容如下：

(1) books.htm

```
<h3>计算机</h3>
<ul>
<li><a href="database.htm">数据库</a></li>
<li><a href="program.htm">程序设计</a></li>
<li><a href="network.htm">计算机网络</a></li>
</ul>
```

(2) database.htm

```
<h3>数据库</h3>
<ul>
<li><a href="mysql.htm">MySQL 数据库</a></li>
</ul>
```

(3) program.htm

```
<h3>程序设计</h3>
<ul>
<li><a href="python.htm">Python 程序设计</a></li>
<li><a href="java.htm">Java 程序设计</a></li>
</ul>
```

(4) network.htm

```
<h3>计算机网络</h3>
```

(5) mysql.htm

```
<h3>MySQL 数据库</h3>
```

(6) python.htm

```
<h3>Python 程序设计</h3>
```

(7) java.htm

```
<h3>Java 程序设计</h3>
```

然后再用 Flask 设计一个 server.py 的 Web 程序来呈现它们：

```
import flask
import os
app=flask.Flask(__name__)

def getFile(fileName):
    data=b""
    if os.path.exists(fileName):
        fobj=open(fileName,"rb")
        data=fobj.read()
        fobj.close()
    return data

@app.route("/")
def index():
    return getFile("books.htm")

@app.route("/<section>")
def process(section):
    data=""
    if section!="":
        data=getFile(section)
    return data

if __name__=="__main__":
    app.run()
```

这个程序的默认网址是 http://127.0.0.1:5000，访问这个网址后执行 index 函数，返回 books.htm 的网页，如图 3-1-1 所示。

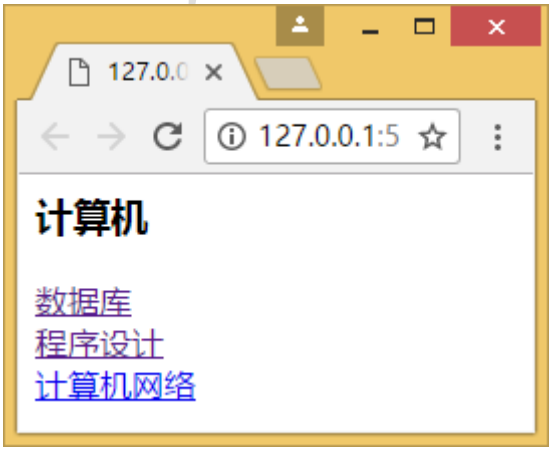

图 3-1-1 Web 网站

点击每个超级链接后会转去数据库、程序设计、计算机网络各个网页，这些网页的结构实际

上是一棵树，如图 3-1-2 所示。

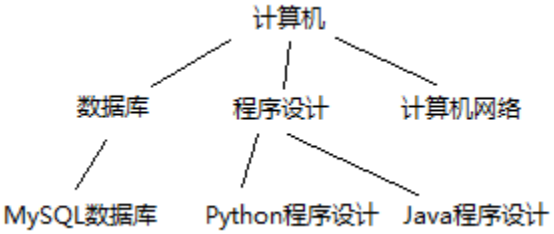

图 3-1-2 Web 网站结构

3.1.2 递归程序爬取数据

现在我们来设计一个客户端程序 client.py 爬取这个网站各个网页的<h3>的标题值，设计的思想如下：

(1) 从 books.htm 出发；
(2) 访问一个网页，获取<h3>标题；
(3) 获取这个网页中所有<a>超级链接的 href 值形成 links 列表；
(4) 循环 links 列表，对于每个链接 link 都指向另外一个网页，递归回到(2)；
(5) 继续 links 的下一个 link，直到遍历所有 link 为止；

因此程序是一个递归调用的过程，程序如下：

```
from bs4 import BeautifulSoup
import urllib.request

def spider(url):
    try:
        data=urllib.request.urlopen(url)
        data=data.read()
        data=data.decode()
        soup=BeautifulSoup(data,"lxml")
        print(soup.find("h3").text)
        links=soup.select("a")
        for link in links:
            href=link["href"]
            url=start_url+"/"+href
            #print(url)
            spider(url)
    except Exception as err:
        print(err)

start_url="http://127.0.0.1:5000"
spider(start_url)
print("The End")
```

执行的结果如下：

```
计算机
```

数据库
MySQL 数据库
程序设计
Python 程序设计
Java 程序设计
计算机网络
The End

如果读者对数据结构熟悉，很显然程序在使用深度优先遍历这棵树，实际上这种递归程序都是采用深度优先的方法遍历树。

3.1.3 深度优先爬取数据

如果我们不使用递归程序实现深度优先的顺序爬取网站数据，也可以设计一个栈 Stack 来完成。在 Python 中实现一个栈十分简单，Python 中的列表 list 就是一个栈，很容易设计自己的一个栈 Stack 类：

```
class Stack:
    def __init__(self):
        self.st=[]
    def pop(self):
        return self.st.pop()
    def push(self,obj):
        self.st.append(obj)
    def empty(self):
        return len(self.st)==0
```

其中 push 是压栈函数、pop 是出栈函数、empty 判断栈是否为空。采用 Stack 类后我们可以设计深度优先的顺序爬取数据的客户端程序色的思想如下：

(1) 第一个 url 入栈；

(2) 如果栈为空程序结束，如不为空出栈一个 url，爬取它的<h3>标题值；

(3) 获取 url 站点的所有超级链接<a>的 href 值，组成链接列表 links，把这些链接全部压栈；

(4) 回到(2)

client.py 如下：

```
from bs4 import BeautifulSoup
import urllib.request

class Stack:
    def __init__(self):
        self.st=[]
    def pop(self):
        return self.st.pop()
    def push(self,obj):
        self.st.append(obj)
    def empty(self):
        return len(self.st)==0
```

```
def spider(url):
    stack=Stack()
    stack.push(url)
    while not stack.empty():
        url=stack.pop()
        try:
            data=urllib.request.urlopen(url)
            data=data.read()
            data=data.decode()
            soup=BeautifulSoup(data,"lxml")
            print(soup.find("h3").text)
            links=soup.select("a")
            for i in range(len(links)-1,-1,-1):
                href=links[i]["href"]
                url=start_url+"/"+href
                stack.push(url)
        except Exception as err:
            print(err)

start_url="http://127.0.0.1:5000"
spider(start_url)
print("The End")
```

程序结果：

```
计算机
数据库
MySQL 数据库
程序设计
Python 程序设计
Java 程序设计
计算机网络
The End
```

3.1.4 广度优先爬取数据

遍历网站树还有一种广度优先的顺序，这要使用到队列，在 Python 中实现一个队列十分简单，Python 中的列表 list 就是一个队列，很容易设计自己的一个队列 Queue 类：

```
class Queue:
    def __init__(self):
        self.st=[]
    def fetch(self):
        return self.st.pop(0)
    def enter(self,obj):
        self.st.append(obj)
```

```
    def empty(self):
        return len(self.st)==0
```

其中 enter 是入列函数、fetch 是出列函数、empty 判断列是否为空。采用 Queue 类后我们可以设计广度优先的顺序爬取数据的客户端程序的思想如下：

(1) 第一个 url 入列；

(2) 如果列空程序结束，如不为空出列一个 url，爬取它的<h3>标题值；

(3) 获取 url 站点的所有超级链接<a>的 href 值，组成链接列表 links，把这些链接全部入栈；

(4) 回到(2)

client.py 程序如下：

```
from bs4 import BeautifulSoup
import urllib.request

class Queue:
    def __init__(self):
        self.st=[]
    def fetch(self):
        return self.st.pop(0)
    def enter(self,obj):
        self.st.append(obj)
    def empty(self):
        return len(self.st)==0

def spider(url):
    queue=Queue()
    queue.enter(url)
    while not queue.empty():
        url=queue.fetch()
        try:
            data=urllib.request.urlopen(url)
            data=data.read()
            data=data.decode()
            soup=BeautifulSoup(data,"lxml")
            print(soup.find("h3").text)
            links=soup.select("a")
            for link in links:
                href=link["href"]
                url=start_url+"/"+href
                queue.enter(url)
        except Exception as err:
            print(err)

start_url="http://127.0.0.1:5000"
spider(start_url)
```

```
print("The End")
```

程序结果：

```
计算机
数据库
程序设计
计算机网络
MySQL 数据库
Python 程序设计
Java 程序设计
The End
```