第14章 网络程序设计

内容

14.1 网络协议

14.2 套接字编程

14.3 Web编程

14.4 典型案例

14.1 网络协议

14.1.1 互联网协议族

14.1.2 TCP/IP

14.1.1 互联网协议族

- ✓ 要把全世界不同类型的计算机都连接起来，必须规定一套全球通用的通信协议，这就是互联网协议簇(Internet Protocol Suite，IPS)。
- ✓ 互联网协议族中最具有代表性的: OSI(Open System Interconnection Reference Model，开放系统互连参考模型)和TCP/IP(Transmission Control Protocol/Internet Protocol，传输控制协议/网际协议)。
- ✓OSI和TCP/IP及对应关系见表14.1 (教材P258)。

OSI简介

- 为使不同计算机厂家生产的计算机能相互通信，以便在更大范围内建立计算机网络，国际标准化组织**(ISO)** 在**1978** 年提出“开放系统互联参考模型”， 即著名的**OSI/RM(Open System Interconnection/Reference Model)**。
- **OSI**参考模型的基本技术是分层技术。按照**ISO 7498**的定义，**OSI**的体系结构具有**7**个层次，如图**7-1**所示。

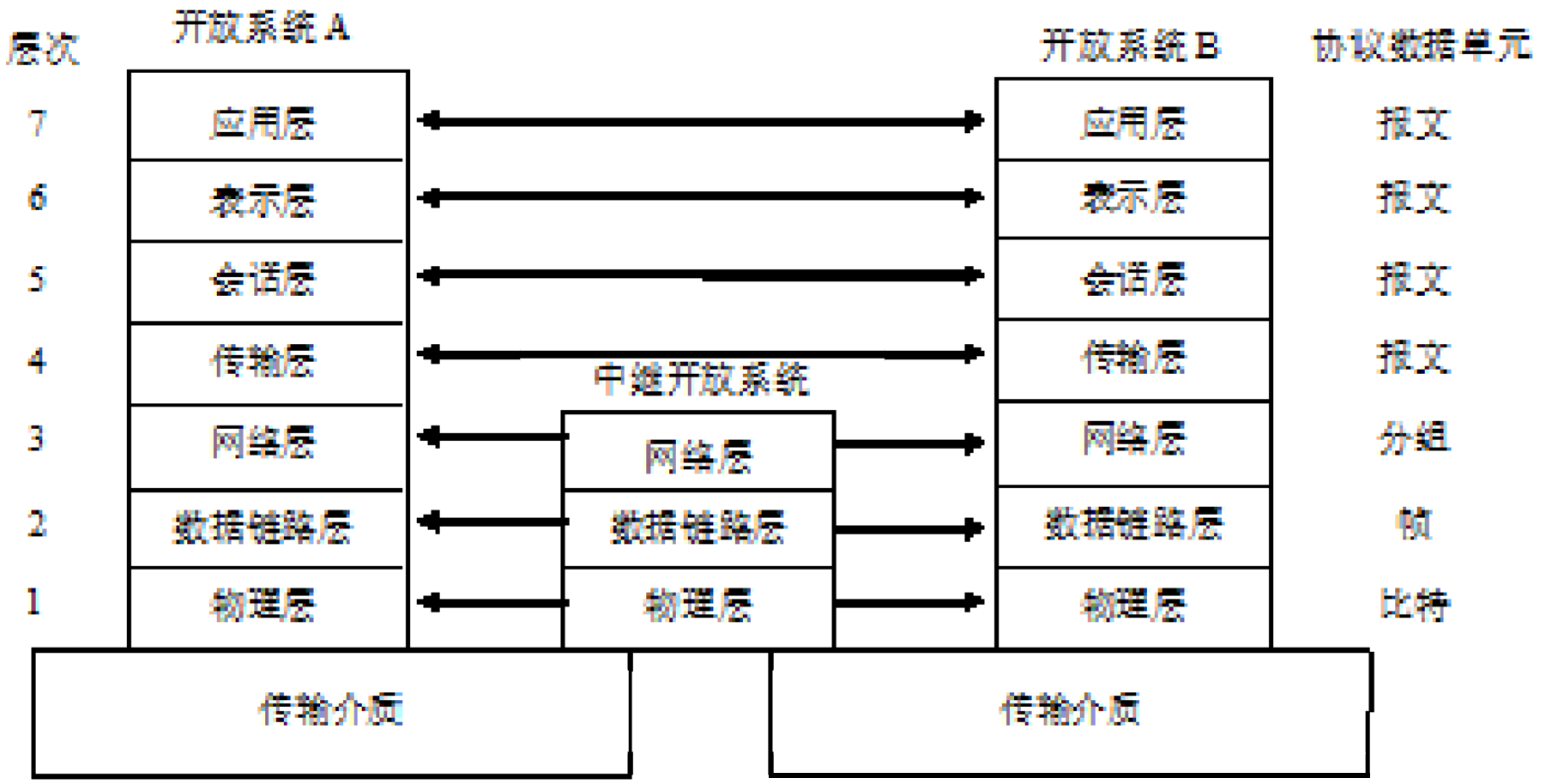

图 OSI的7层体系结构

TCP/IP 简介

- **TCP/IP**是一种网际互联通信协议，其目的在于通过它实现网际间各种异构网络和异种计算机的互连通信，因特网采用的即为**TCP/IP**协议。
- **TCP/IP**协议的核心思想是对于**ISO 7**层协议，把千差万别的底层的两层协议**(**物理层和数据链路层**)**有关部分称为物理网络，而在传输层和网络层建立一个统一的虚拟逻辑网络，以这样的方法来屏蔽或隔离所有物理网络的硬件差异，从而实现普遍的连通性。
- **TCP/IP**协议族把整个协议分成**4**个层次，如图所示：

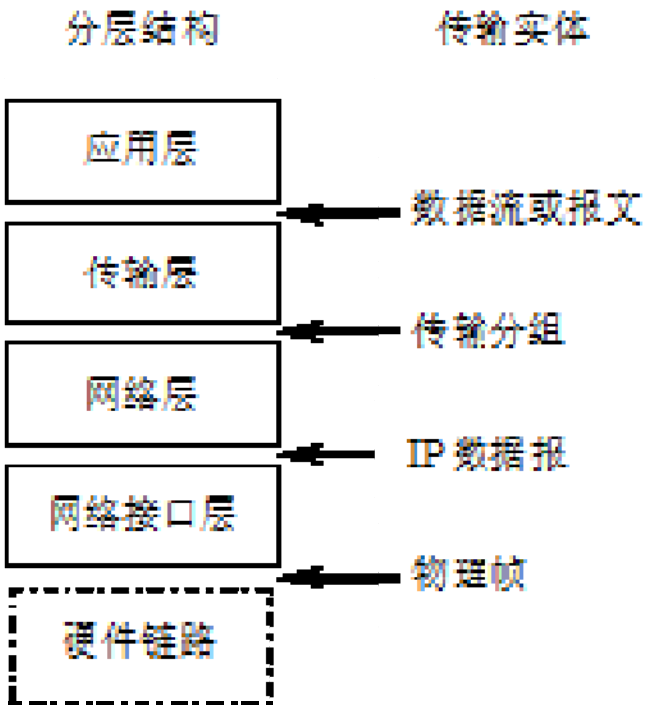

图 **TCP/IP** 协议模型

TCP/IP与OSI的关系

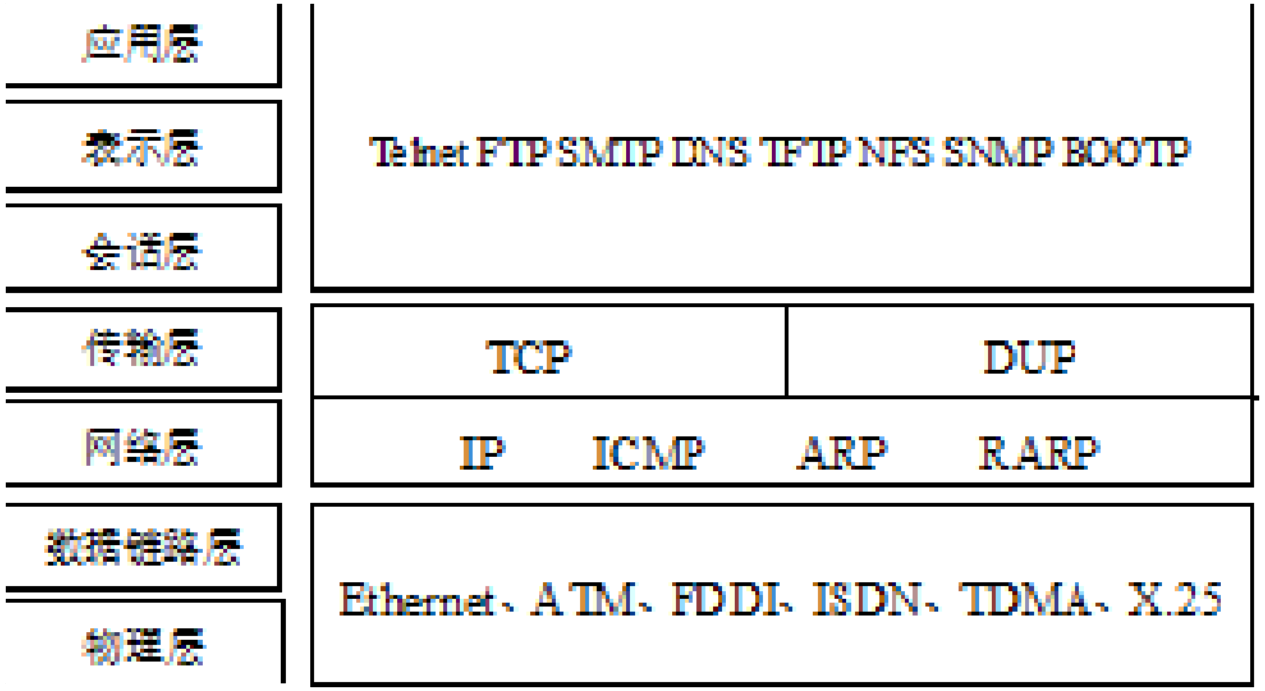

图 TCP/IP与OSI的关系

TCP/IP协议族

- TCP/IP中包含了一组通信协议，而被称为协议族。TCP/IP协议族中包括上百个互为关联的协议，不同功能的协议分布在不同的协议层。下面介绍几个常用协议。
- (1)网络互联协议(Internet Protocol, IP)
- IP协议是TCP/IP协议体系中最重要的部分，是一个无连接的协议，位于第三层的网络层。IP协议的基本任务是采用数据报方式，通过因特网传送数据，将源主机的报文分组发送到目的主机。
- (2)传输控制协议(Transmission Control Protocol, TCP)
- TCP/IP的传输层有两个重要的协议：一个是面向连接的传输控制协议(TCP)，另一个是无连接的用户数据报协议(User Datagram Protocol, UDP)，这两个协议位于TCP/IP参考模型的第三层传输层，主要负责应用程序之间实现端到端的通信。TCP协议和UDP协议都使用IP协议，这两个协议在发送数据时，其协议数据单元都作为下层IP数据报中的数据

- **(3)用户数据报协议(User Datagram Protocol, UDP)UDP协议也是建立在IP协议之上，同IP协议一样提供无连接数据报传输。UDP本身并不提供可靠性服务，相对IP协议，唯一增加的能力是提供协议端口，以保证进程通信。虽然UDP不可靠，但效率很高。在不需要TCP全部服务时，有时用UDP取代TCP。例如，简单文件传输协议(TFTP)、简单网络管理协议(SNMP)和远过程调用(RPC)就使用了UDP。**
- **(4)网路终端协议(Telecommunication Network, Telnet)**
- 远程终端访问协议提供一种非常广泛、双向的、8字节的通信功能，位于应用层。这个协议提供了一种与终端设备或终端进程交互的标准方法。该协议提供的最常用的功能是远程登录。远程登录就是通过因特网进入和使用远距离的计算机系统，就像使用本地计算机一样。远程计算机可以在同一间屋子里或同一校园内，也可以在数千公里之外

- **(5)文件传输协议(File Transfer Protocol, FTP)**
- 文件传输协议用于控制两个主机之间的文件交换。**FTP**是**Internet**上最早使用的文件传输程序，同**Telnet**一样，使用户能够登录到**Internet**的一台远程计算机，把其中的文件传送回自己的计算机系统，或者反过来，把本地计算机上的文件传送到远程计算机系统。
- **(6)简单邮件传送协议(Simple Mail Transfer Protocol, SMTP)**
- **Internet**标准中的电子邮件是一个简单的面向文本的协议，用来有效、可靠地传送邮件。作为应用层协议，**SMTP**不关心下层采用什么样的传输服务。它通过**TCP**连接来传送邮件。

- **(7)域名服务(Domain Name Service, DNS)**
- **DNS是一个名字服务协议，提供了名字到IP地址的转换。虽然DNS的最初目的是使邮件的发送方知道邮件接收主机及邮件发送主机的IP地址，但现在它的用途越来越广。**
- **(8)超文本传输协议(Hyper Text Transfer Protocol, HTTP)**
- **HTTP是网络上应用最广泛的一种协议。所有的WWW文件都必须遵守这个标准，设计HTTP最初的目的是为了提供一种发布和接收HTML页面的方法。**

14.1.2 TCP/IP

(1)TCP/IP中最重要的两个协议是TCP和IP。

(2)IP负责把数据从一台计算机通过网络发送到另一台计算机上。

✓由于网络链路复杂，两台计算机之间经常有多条线路。

✓路由器负责决定如何把一个IP包转发出去。

✓IP包的特点是按块发送，途经多条路由，但不保证能到达，也不保证按顺序到达。

✓同一台计算机上运行着多个网络程序，每个网络程序都向操作系统申请唯一的端口号。

✓两台计算机上的网络程序要建立网络连接就需要具有各自的IP地址和端口(Port)号。

✓一个IP包除包含要传输的数据外，还包含源IP地址和目标IP地址、源端口和目标端口。

14.1.2 TCP/IP

(3)TCP建立在IP之上，负责在两台计算机之间建立可靠的连接，保证数据包按顺序到达。

✓TCP通过握手(Handshake)建立连接，然后对每个IP包编号，确保对方按顺序收到。如果包丢掉了，就自动重发。

✓许多常用的高级协议都建立在TCP的基础上，如用于浏览网页的HTTP、发送邮件的SMTP等协议。

(4)UDP(User Datagram Protocol，用户数据报协议)与TCP一样用于处理数据包，是一种无连接的协议。

✓UDP提供数据包分组、组装，但不能对数据包进行排序。

✓UDP用来支持那些需要在计算机之间传输数据的网络应用，主要作用是将网络数据压缩成数据包的形式。

14.2 套接字编程

14.2.1 套接字简介

14.2.2 基于TCP的套接字编程

14.2.3 基于UDP的套接字编程

14.2.1 套接字简介

- ✓ 什么是Socket：套接字(socket)是基于TCP/IP的一个封装的网络组件，可以理解为两个端点的应用程序之间的“信息通道”。
- ✓ 套接字规定了通信所用的协议簇，明确了通信双方的IP地址和端口号。通过网络连接的计算机通过套接字可以相互发送消息。套接字分为服务器套接字和客户机套接字。应用程序通常通过套接字向网络发出请求或者应答网络请求。 套接字是一个软件抽象层，不负责发送数据，真正发送数据的是套接字后面的协议。
- ✓ 套接字有B/S架构和C/S架构这两种，本质上都是客户端和服务端之间的数据通信。

- ✓ 在Python中，一个套接字就是socket模块中的socket类的一个实例，创建套接字一般格式如下：
  - ✓ socket.socket([AddressFamily [, type [,protocol]]])
- ✓ 其中，AddressFamily: 代表套接字家族，可以使AF_UNIX或者默认AF_INET，type: 套接字类型，根据面向连接或非连接分为默认的SOCK_STREAM或SOCK_DGRAM，protocol: 一般不填、默认为0。
- ✓ 套接字函数分为服务器端套接字函数(见表14.2)、客户端套接字函数(见表14.3)、公共套接字函数(见表14.4) (教材P259) 。

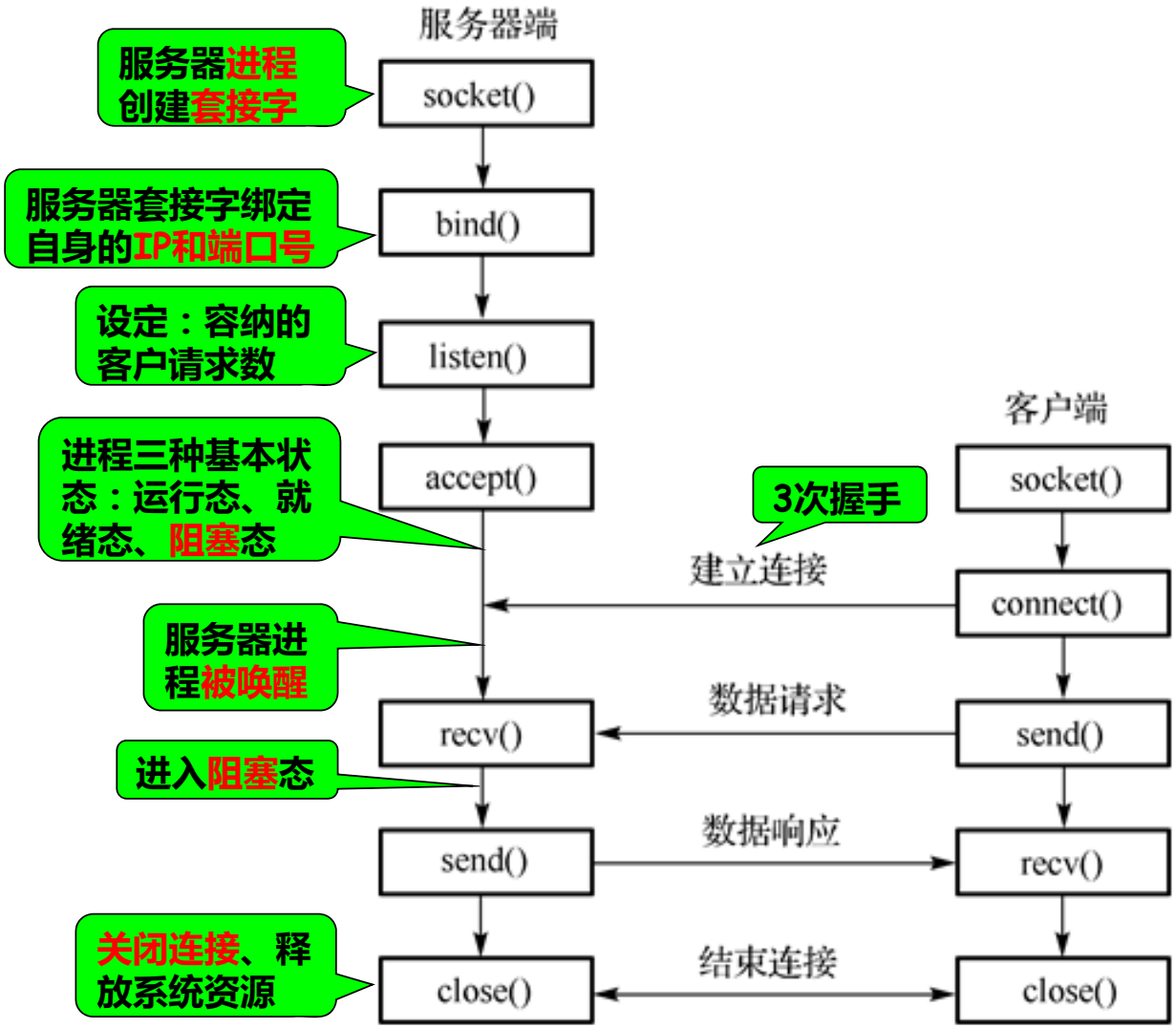

图 14.1　基于 TCP 的套接字编程模型

14.2.2 基于TCP的套接字编程

1．服务器端

服务器端实现步骤如下。

(1) 创建套接字：使用serverSocket=socket.socket( )函数完成。

(2) 绑定本地信息：通过serverSocket.bind((IP, port))函数完成。#该套接字仅含本地信息

(3) 定义最多可容纳的等待接收的传入连接数：通过serverSocket.listen(n)函数完成。

(4) 进入等待状态：使用conn, addr = serverSocket.accept()函数完成，等待客户发起连接请求，服务器进程进入阻塞状态。

进程三种基本状态：运行态、就绪态、阻塞态

一旦客户端请求抵达，服务器被唤醒：

conn：新的套接字对象，增加了远程端点信息(addr)，用于一次通信过程；

addr：客户端地址(二元组)

(5) 和客户端通信：通过send()、recv()函数和客户端通信。相关进程同样可能被阻塞。

(6) 关闭连接：通过连接对象的close()函数关闭连接。

14.2.2 基于TCP的套接字编程

2．客户端

客户端实现步骤如下。

(1) 创建socket对象：使用clientSocket = socket.socket()函数完成。

(2) 客户端socket对象连接服务端：通过clientSocket.connect(IP, port)函数连接服务器。客户端connect()方法使用的地址为：服务器的信息。

(3) 和客户端通信：通过send()、recv()函数和服务器端通信。

(4) 关闭套接字：通过clientSocket.close()函数关闭套接字。

例：基于TCP的套接字编程

【例14.1】基于TCP套接字的服务器-客户端简单通信。服务器端程序代码：

```
import socket
serverSocket = socket.socket()                    #创建套接字对象.
serverSocket.bind(('127.0.0.1',12345))            #绑定
serverSocket.listen(5)              #设置挂起的连接数，顺序处理每一个客户端请求！
print("等待客户端发起连接请求！ ")
while True:     #本例中，服务器始终开启服务：依次处理每一个客户端请求
  conn,addr = serverSocket.accept()               #等待客户端连接请求、阻塞！！
  #... …等待客户端请求，之后被唤醒
  print("客户端连接成功...")
#print( conn )   #<socket.socket fd=372, family=AddressFamily.AF_INET,
type=SocketKind.SOCK_STREAM, proto=0, laddr=('127.0.0.1', 12345), raddr=('127.0.0.1', 55739)>
#l:local r:remote
#print( addr )   #('127.0.0.1', 55739)
```

14.2.2 基于TCP的套接字编程

```
while True:        #(嵌套)循环处理一个客户端的多次请求，位于上层的while True内部
    try:
      recv_data = conn.recv(1024)                    #等待客户端消息、阻塞！！.
      if len(recv_data) == 0:                        #客户端终止通信( 客户端.close)
        print("服务器: 好吧，我也退出连接！")
        break                                        #退出循环，即将断开本次通信连接
      print(“客户端：”+ str(recv_data.decode()))     #字节码(默认UTF-8)变Unicode
      send_data = "(阿甘)妈妈常常说，人生就如同一盒巧克力，你永远无法知道下一粒你会拿到什么"
      conn.send(bytes(send_data, encoding='utf8'))   #给客户端发送信息. Unicode变字节码(UTF-8)
      print("服务器:" + send_data)
    except Exception:
      break
conn.close()       #关闭连接.终止一个客户连接。返回上一层循环：接收下一个请求
```

14.2.2 基于TCP的套接字编程

客户端程序代码：

```
import socket
clientSocket = socket.socket()                  #创建套接字.
clientSocket.connect(('127.0.0.1',12345))       #绑定服务器端的套接字.
while True:
  send_data = input("客户端: ").strip()          #从键盘输入信息.
  if send_data == 'exit':
    print("客户端: 我要退出连接了！ ")
    break
  if len(send_data) == 0:                       #避免：空消息导致通信错误终止
    continue
  clientSocket.send( bytes(send_data ) )        #发送信息. Unicode变字节码(默认UTF-8)
  recv_data = clientSocket.recv(1024)           #接收服务器信息.
  print("服务器： " + str( recv_data.decode()))      #输出信息.
#input( 'test' )              #验证：通信终止源于下面的.close( )
clientSocket.close()                            #关闭连接.
```

14.2.2 基于TCP的套接字编程

运行结果(服务器端-先运行):
等待客户端发起连接请求!
客户端连接成功...
客户端：你知道《阿甘正传》中最经典的一句台词是什么吗?
服务器: (阿甘)妈妈常常说，人生就如同一盒巧克力，你永远无法知道下一粒你会拿到什么。
服务器: 好吧，我也退出连接!

运行结果(客户端-后运行):
客户端: 你知道《阿甘正传》中最经典的一句台词是什么吗? （键盘输入）
服务器：(阿甘)妈妈常常说，人生就如同一盒巧克力，你永远无法知道下一粒你会拿到什么。
客户端: **exit**
客户端: 我要退出连接了!

```python
import socket
serverSocket = socket.socket()                  #创建套接字.
serverSocket.bind(('127.0.0.1',12345))  #绑定套接字,指定端口号：12345
serverSocket.listen( 5 )                        #设置挂起的连接数
while True:                             #永远接受客户端请求
  print("等待下一个客户端的连接请求... ...")
  conn,addr = serverSocket.accept()     #等待客户端连接请求、阻塞
  print();
  print('连接成功，客户端信息是：');print( conn );print( addr )
  while True:       #保持与客户端的连接，直至客户端终止
    try:
      print("等待客户端的消息：")
      recv_data = conn.recv(1024)       #接收信息、阻塞
      if len(recv_data) == 0:
        print("服务器:客户端结束通信、本次连接终止！")
        break
      print("客户端的消息是：" + str( recv_data.decode() ) )
      while True:     #忽略服务器端的空白行，否则：可能形成通信的死锁
        send_data = input("我是服务器端，反馈给客户端的消息是：")
        if len(send_data) > 0:break
      conn.send(bytes(send_data,encoding='utf8')) #向客户端发送信息.
    except Exception:
      print('通信异常... ...')
      break
conn.close()    #关闭连接.
print ('服务器关闭所有结束！')
```

```python
import socket
clientSocket = socket.socket()              #创建套接字.
clientSocket.connect(('127.0.0.1',12345)) #服务器的IP、端口号
print("连接成功... ...")

while True:
  send_data = input("我是客户端，请输入消息（exit退出）：").strip()
  if send_data == 'exit':
    print("客户端: 我要退出连接了！")
    break
  if len(send_data) == 0:
    continue
  clientSocket.send( bytes(send_data,encoding='utf8') ) #发送信息.
  print("客户端已发送，等候服务器反馈...")
  recv_data = clientSocket.recv(1024)                   #等待服务器信息.
  print("服务器的反馈是：" + str(recv_data.decode()))   #输出服务器信息.
clientSocket.close()
print("\n我是客户端:通信结束！")
```

14.2.3 基于UDP的套接字编程

- ✓ TCP建立可靠连接，并且通信双方都可以以流的形式发送数据。
- ✓ UDP则是面向无连接的协议。使用UDP时，不需要建立连接，只需要知道对方的IP地址和端口号，就可以直接发数据包，但不能保证到达。

- ✓ 用UDP传输数据不可靠，优点是比TCP的速度快。
- ✓ 对于不要求可靠到达的数据，可以使用UDP。
- ✓ 和TCP类似，使用UDP的通信双方也分为客户端和服务器端。
- ✓ 基于UDP的套接字编程见图14.2。

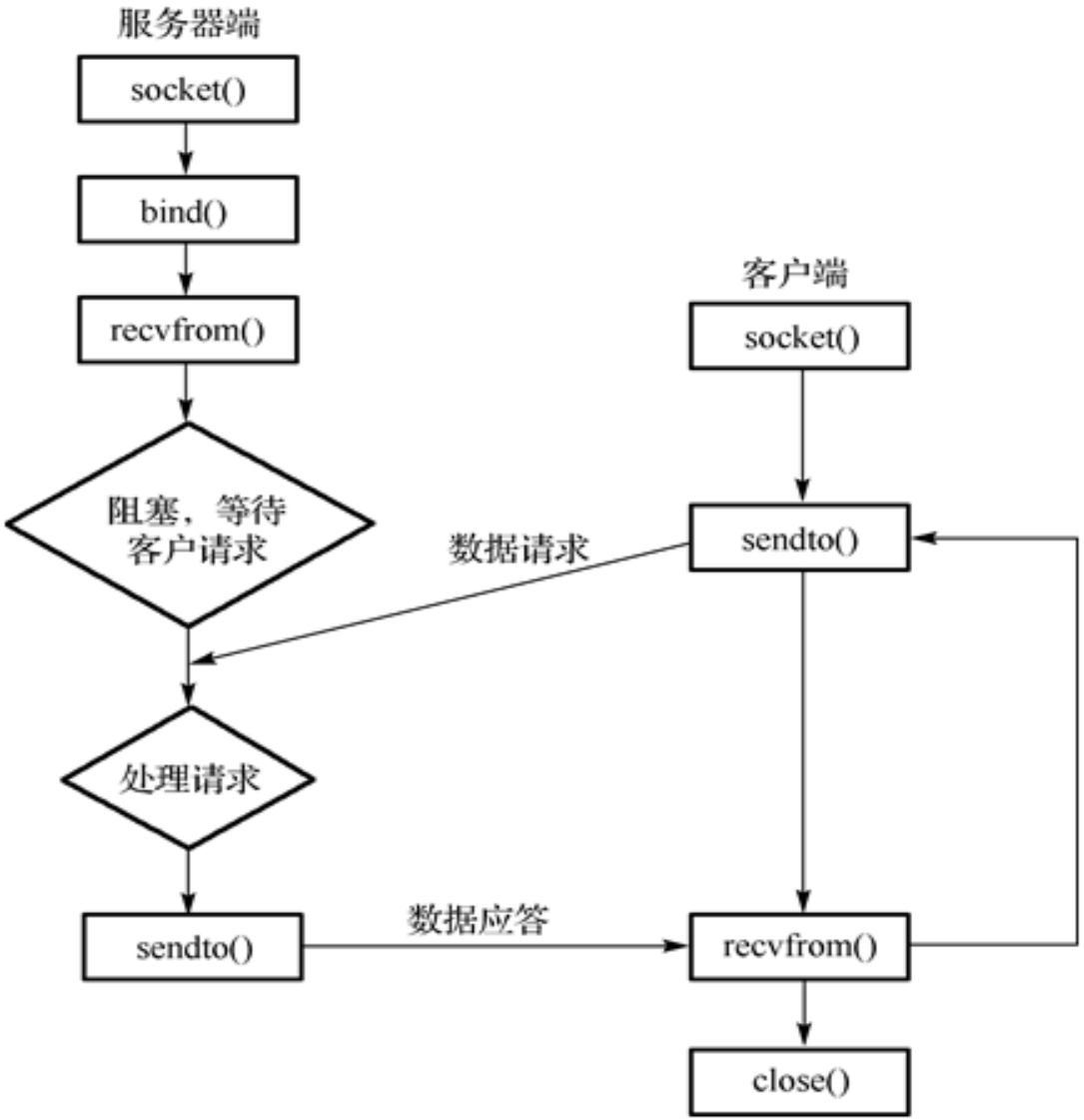

图 14.2　基于 UDP 的套接字编程

14.2.3 基于UDP的套接字编程

1．服务器端

(1) 创建套接字：使用serverSocket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)函数完成。

(2) 绑定套接字：通过serverSocket.bind((IP, Port)) 函数完成，但不需要调用listen()方法，而是直接接收来自任何客户端的数据。

(3) 接收数据：使用data, addr = serverSocket.recvfrom(1024)函数完成，接收来自客户端的数据。

(4) 发送数据：使用sendto()函数完成，把数据发送给客户端。

14.2.3 基于UDP的套接字编程

2．客户端

(1) 创建套接字：使用clientSocket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)函数完成。

(2) 发送数据：无需connect()，通过clientSocket.sendto(data, (IP, Port))函数完成，向服务器发数据。

(3) 接收数据：使用data, addr = clientSocket.recvfrom(1024)函数完成，接收来自服务器的数据。

(4) 关闭连接：通过clientSocket.close()函数关闭套接字。

14.2.3 基于UDP的套接字编程

【例14.2】 基于UDP套接字的服务器-客户端简单通信。服务器端程序代码：

```
import socket
serverSocket = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)      #创建套接字.
serverSocket.bind(('127.0.0.1',12345))                              #绑定套接字.
print("在12345端口绑定UDP...")
answers = ["伽利略,哥白尼","爱迪生,特斯拉","诺贝尔,门捷列夫"]        #回答列表.
while True:
  i = 0              #bug：永远响应第一个元素
  #input('回车！之后等待接收、阻塞')
  question,addr = serverSocket.recvfrom(1024)     #接收客户端信息.阻塞！
  #input('已经收到消息、被唤醒。回车继续！')
  print("客户端问:",question.decode())              #输出接收的问题.
  #发送数据给客户端.
  serverSocket.sendto(bytes("服务器答:" + answers[i],encoding='utf8'),addr)
  i = i + 1
serverSocket.close()       #关闭套接字.
```

14.2.3 基于UDP的套接字编程

客户端程序代码：

```
import socket
clientSocket = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)      #创建套接字对象.
questions = ['世界上最伟大的天文学家是谁？','世界上最伟大的发明家是谁？',
             '世界上最伟大的化学家是谁？']                          #问题列表.
#遍历发送列表questions中的问题.
for question in questions:
  #发送问题给服务器.
  clientSocket.sendto(bytes(question, encoding='utf8'), ('127.0.0.1',12345) )
  print("客户端问:" + question)                  #输出发送的问题.
  answer,addr = clientSocket.recvfrom(1024)        #接收服务器回答.
  print( str( answer.decode() ) )                  #输出回答.str多余
clientSocket.close()                             #关闭套接字.
```

14.2.3 基于UDP的套接字编程

运行结果(服务器端-先运行):

在12345端口绑定UDP...

客户端问: 世界上最伟大的天文学家是谁？

客户端问: 世界上最伟大的发明家是谁？

客户端问: 世界上最伟大的化学家是谁？

运行结果(客户端-后运行):

客户端问: 世界上最伟大的天文学家是谁？

服务器答: 伽利略, 哥白尼

客户端问: 世界上最伟大的发明家是谁？

服务器答: 爱迪生, 特斯拉

客户端问: 世界上最伟大的化学家是谁？

服务器答: 诺贝尔, 门捷列夫

理解

- **Python** 提供了两个级别访问的网络服务：
- ①低级的网络服务支持基本的 **Socket**，提供了标准的**BSD Sockets API**，可以访问底层操作系统**Socket**接口的全部方法。
- **Socket**工作于应用程序的应用层和传输层之间。
- 使用**socket**一次只能和一个客户端进行交互，其余客户端通行则需排队等候（等候先前客户断开连接）。

- ②高级的网络服务模块**SocketServer**，提供了服务器中心类，能够同时连接多个客户端、从而简化网络服务器的开发。
- 主要区别在于服务端的不同。

例： SocketServer编程

```
import socketserver
class MyTCPHandler( socketserver.BaseRequestHandler):
    def handle(self):
        while True:
            try:
                self.data = self.request.recv(1024).strip()
                print("{} wrote:".format(self.client_address[0]))
                print(self.data)
                self.request.send(self.data.upper())
            except ConnectionResetError as e:
                print("err",e)
                break
if __name__ == "__main__":
    HOST,PORT = "localhost",9999
    server = socketserver.ThreadingTCPServer((HOST,PORT),MyTCPHandler)
    server.serve_forever()              #能连接多个客户端
    server.handle_request()             #只能连接一个客户端，其余的客户端连接的话会被拒绝
```

14.3 Web编程

14.3.1 Web编程简介

14.3.2 基于Django的Web编程

14.3.1 Web编程简介

✓ 随着互联网和移动互联网的兴起，浏览器/服务器(Browser/Server，B/S)架构得到了快速发展。

✓ B/S架构下的应用程序逻辑和数据都存储在服务器端，客户端只需要安装浏览器，然后浏览器通过Web Server与后台数据库进行数据交互。

✓ B/S架构的工作流程如下：

(1) 客户端发送请求。

(2) 服务器端处理请求。

(3) 服务器端发送响应。

(4) 浏览器显示结果。

14.3.1 Web编程简介

- ✓ 为了加快Web开发的效率，减轻网页开发的工作负荷，提升代码的可重用性，通常使用Web应用框架(Web Application Framework)进行Web开发。
- ✓ Web应用框架是一种开发框架，用来支持动态网站、网络应用程序及网络服务的开发。
- ✓ 目前已经有十多种基于Python的Web应用框架，如Django、Flask、Web2py、Tornado和Bottle等。
- ✓ 下面对几种常用、基于Python的Web应用框架进行简单介绍。

14.3.2 基于Django的Web编程

1．简介

✓ Django是一个开放源代码的Web应用框架，由Python写成。

✓ Django 项目源自一个在线新闻Web站点，诞生于2003年，于2005年7月在获得BSD许可证后发布。

✓ Django框架的核心控件如下。

(1) 用于创建模型的对象关系映射(Object Relational Mapping，ORM)。。

(2) 自动化管理界面。

(3) URL设计。

(4) 模板系统。

(5) 缓存系统。

14.3.2 基于Django的Web编程

Django对传统的MVC设计模式进行了修改：

- ✓ 将视图进一步分成视图(View)和模板(Template)两部分，将动态的逻辑处理与静态的页面展现分离开；
- ✓ 模型采用ORM技术，将关系型数据库表抽象成面向对象的Python类，将表操作转换成类操作，从而避免了复杂的SQL语句编写。
- ✓ 通过这种修改，形成了Django独特的MTV模式，即模型、模板和视图。
- ✓ 在MTV中，模型和MVC中的定义相同，模板负责将数据与HTML语言结合，视图负责实际的业务逻辑实现。

14.3.2 基于Django的Web编程

2．下载与安装

✓ 因为Django的不同版本之间有差异，所以Django支持的Python版本也各不相同。

✓ 本书案例使用的是与Python 3.7.2版本对应的Django 2.2。

✓ Django的下载网址为https://www.djangoproject.com/ download/。

✓ 考虑到集成开发环境的快捷性和方便性，本书中关于Django的案例基于PyCharm。

✓ Django在PyCharm中的下载和安装方法参见1.2.5节。

14.3.2 基于Django的Web编程

3．创建Djanogo项目

【例14.3】创建一个Django项目，通过浏览器访问开发服务器，在浏览器上显示“Hello，world!”。

步骤如下。

(1) 创建Django项目

✓ 打开“New Project”(创建新项目)对话框，选择创建Django项目类型，选择项目存储路径，输入项目名称DjangoWeb和应用名称FirstWeb，然后单击“Create”按钮创建项目(见图14.3)。

✓ 创建成功后的Django项目目录和文件结构见图14.4。

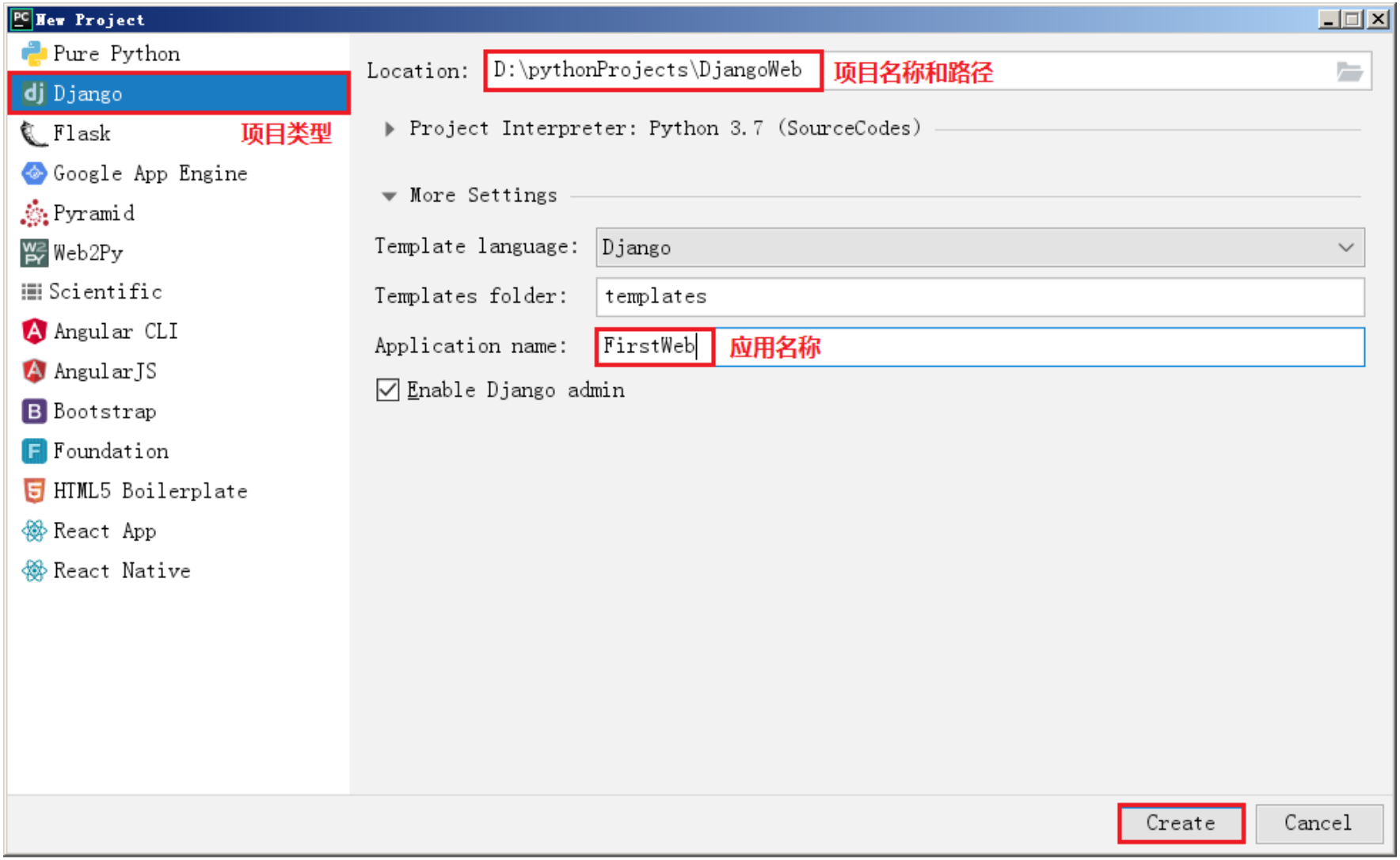

图14.3 “New Project”对话框

图14.4 Django项目目录和文件结构

14.3.2 基于Django的Web编程

(2) 修改路由文件

路由文件urls.py中的程序代码如下：

```
from django.contrib import admin
from django.urls import path
from FirstWeb import views
urlpatterns = [
  path('admin/',admin.site.urls),
  path(r'',views.index),
]
```

(3) 修改配置文件

配置文件views.py中的程序代码如下：

```
from django.shortcuts import render
def index(request):
  return render(request,'index.html')
```

14.3.2 基于Django的Web编程

(4) 创建网页文件

在templates文件夹下创建一个名为index.html的网页文件，代码如下：

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>第一个Django程序</title>
  </head>
  <body>
Hello, world!
</body>
</html>
```

14.3.2 基于Django的Web编程

运行开发服务器，在浏览器中输入**http://127.0.0.1:8000**，运行结果见图**14.5**。

图 14.5　运行结果

14.3.2 基于Django的Web编程

4．表单

【例14.4】 在**Django**项目中使用表单，项目名为**DjangoWeb**，应用名为**FirstWeb**。

(1) 修改路由文件

路由文件**urls.py**的内容与**【例14.3】**中的相同。

(2) 修改配置文件

配置文件**views.py**中的代码如下：

```
from django.shortcuts import render
#定义一个数据列表.
list = [{"username":'django','password':'django'}]
```

14.3.2 基于Django的Web编程

```
def index(request):
  #返回前端post过来的用户名和密码.
  username = request.POST.get('username',None)
  password = request.POST.get('password',None)
  #把用户和密码组装成字典.
  if username and password:
    data = {'username':username,'password':password}
    list.append(data)
  return render(request,'index.html',{'form':list})
```

14.3.2 基于Django的Web编程

(3) 创建网页文件

网页文件index.html中的关键代码如下：

```
<body>
  <table border="1">
    <form action="/" method="post"> {% csrf_token %}
      <P><label >用户名:</label><input type="text" name ='username'/></P>
      <P><label>密码：</label><input type="text" name='password'/></P>
      <p><input type="submit" value="提交"/></p>
    </form>
```

14.3.2 基于Django的Web编程

网页文件index.html中的关键代码如下：

```
<body>
  <table border="1">
    <form action="/" method="post"> {% csrf_token %}
      <P><label >用户名:</label><input type="text" name ='username'/></P>
      <P><label>密码：</label><input type="text" name='password'/></P>
      <p><input type="submit" value="提交"/></p>
    </form>
```

14.3.2 基于Django的Web编程

(4) 运行

运行结果见图**14.6**。

图 14.6　表单运行结果

14.3.2 基于Django的Web编程

5. 创建数据模型和迁移数据库

✓ **Django**对各种数据库(包括**Access**、**MySQL**等)提供了很好的支持。

✓ **Django**为这些数据库提供了统一的调用**API**。

1) 创建数据模型

(1) 创建数据模型的常用字段如下。**(**更多字段说明参见教材**P267)**

① **AutoField**：自动增长的**IntegerField**，不指定时会自动创建字属性名为**id**的自动增长属性。

② **BooleanField**：布尔字段，值为**True**或**False**。

③ **NullBooleanField**：支持**Null**、**True**、**False**三种值。

④ **CharField(max_length=**字符长度**)**：字符字段。。

14.3.2 基于Django的Web编程

(2) 通过选项实现对字段的约束，选项如下。**(**更多选项说明参见教材**P268)**

① **null**：如果为**True**，则表示允许为空，默认值是**False**。

② **blank**：如果为**True**，则该字段允许为空白，默认值是**False**。

③ **db_column**：字段的名称。如果未指定，则使用属性的名称。

④ **db_index**：若值为**True,** 则在表中会为此字段创建索引。

⑤ **default**：默认值。

(3) 在模型类中定义**Meta**类，用于设置元信息。常用元选择如下。

① **db_table**：定义该**model**在数据库中的表名称，如**db_table='Students'**。

② **verbose_name**：指明一个易于理解和表述的对象名称，如**verbose_name="**学生**"**。

③ **verbose_name_plural**：对象的复数表述名，如**verbose_name_plural="**学生**"**。

14.3.2 基于Django的Web编程

(4) 不同模型之间的关系有如下三种。

① 一对多关系：外键约束，定义在多类中。

② 多对多关系：定义在两个类中的任意一个。

③ 一对一关系：定义在两个类中的任意一个。

2) 迁移数据库

迁移数据库使用以下两个命令。

① **makemigrations**：在当前目录下生成一个**migrations**文件夹。

② **migrate**：在执行之前生成的**migrations**文件。

14.3.2 基于Django的Web编程

【例14.5】 在Django项目中创建和访问员工数据库，项目名为DjangoWeb，应用名为FirstWeb。

(1) 创建数据模型。

数据模型在models.py文件中创建，其程序代码如下：

```
from django.db import models
class Employee(models.Model):
  eid = models.AutoField(primary_key=True,verbose_name='编号')
  name = models.CharField(max_length=50,verbose_name='姓名')
  age = models.IntegerField(blank=True,null=True,verbose_name='年龄')
  department = models.CharField(max_length=50,verbose_name='部门')
  salary = models.FloatField(blank=True,null=True,verbose_name='年龄')
```

14.3.2 基于Django的Web编程

(2) 迁移数据库。

✓ 选择“Tools”→“Run manage.py Task…”菜单项打开manage.py 控制界面，执行如下命令：

manage.py@chapter14_5 > makemigrations

manage.py@chapter14_5> migrate

✓ 命令运行成功后，在DjangoWeb项目下生成名为db.sqlite3数据库。

✓ 在db.sqlite3数据库中包含一个名为FirstWeb_employee(eid,name,age,department,salary)的数据表。

(3) 修改路由文件。

路由文件urls.py的内容与【例14.3】中的相同。

14.3.2 基于Django的Web编程

(4) 修改配置文件。

配置文件views.py中的程序代码如下：

```
from django.shortcuts import render
from .models import Employee
def index(request):
  SaveData()
  results = Employee.objects.all()
  return render(request,'index.html',{'results':results})
```

14.3.2 基于Django的Web编程

```
#向数据表中添加数据.
def SaveData():
  data1 = Employee(eid='101',name='袭人',age=18,department='行政部',salary=1500)
  data1.save()
  data2 = Employee(eid='102',name='晴雯',age=18,department='技术部',salary=1000)
  data2.save()
  data3 = Employee(eid='103',name='麝月',age=18,department='服务部',salary=1000)
  data3.save()
  data4 = Employee(eid='104',name='秋纹',age=18,department='服务部',salary=1000)
  data4.save()
```

14.3.2 基于Django的Web编程

(5) 修改网页文件。

网页文件index.html中的关键代码如下：

```
<body>
  <table border="">
    <tr>
      <td style="text-align:center;width:60px">编号</td>
      <td style="text-align:center;width:60px">姓名</td>
      <td style="text-align:center;width:60px">年龄</td>
      <td style="text-align:center;width:60px">部门</td>
      <td style="text-align:center;width:60px">薪水</td>
    </tr>
```

14.3.2 基于Django的Web编程

```
    {% for item in results %}
    <tr>
      <td style="text-align:center;width:60px">{{ item.eid }}</td>
      <td style="text-align:center;width:60px">{{ item.name }}</td>
      <td style="text-align:center;width:60px">{{ item.age }}</td>
      <td style="text-align:center;width:60px">{{ item.department }}</td>
      <td style="text-align:center;width:60px">{{ item.salary }}</td>
    </tr>
    {% endfor %}
  </table>
</body>
```

14.3.2 基于Django的Web编程

运行结果见图14.7。

编号	姓名	年龄	部门	薪水
101	袭人	18	行政部	1500.0
102	晴雯	18	技术部	1000.0
103	麝月	18	服务部	1000.0
104	秋纹	18	服务部	1000.0

图 14.7　运行结果

14.3.2 基于Django的Web编程

6．后台管理工具

【例14.6】对Django项目的后台进行管理，项目名为DjangoWeb，应用名为FirstWeb。

(1) 修改路由文件。

路由文件urls.py的内容与【例14.3】中的相同。

(2) 修改配置文件。

配置文件views.py的内容与【例14.5】中的相同。

(3) 创建超级用户。

在登录之前需要先创建超级用户。

14.3.2 基于Django的Web编程

创建超级用户的方法如下。

① 打开**manage.py** 控制界面。

② 按照界面提示输入超级用户名、密码和邮箱等创建超级用户。

manage.py@Django > createsuperuser

Tracking file by folder pattern: migrations

Username (leave blank to use 'administrator'): admin

Email address: 88668866@qq.com

Warning: Password input may be echoed.

Password: admin123456

Superuser created successfully.

创建的超级用户名为**admin**，密码为**admin123456**，邮箱为**88668866@qq.com**。

14.3.2 基于Django的Web编程

(4) 使用管理工具。管理文件FirstWeb/admin.py的程序代码如下:

```
from django.contrib import admin
from FirstWeb.models import Employee
admin.site.register(Employee)
```

✓ 启动开发服务器，然后在浏览器中访问http://127.0.0.1:8000/ admin/，打开后台管理登录界面(见图14.8)。

✓ 输入用户名和密码，登录开发服务器对后台数据进行管理(见图14.9)。

14.3.2 基于Django的Web编程

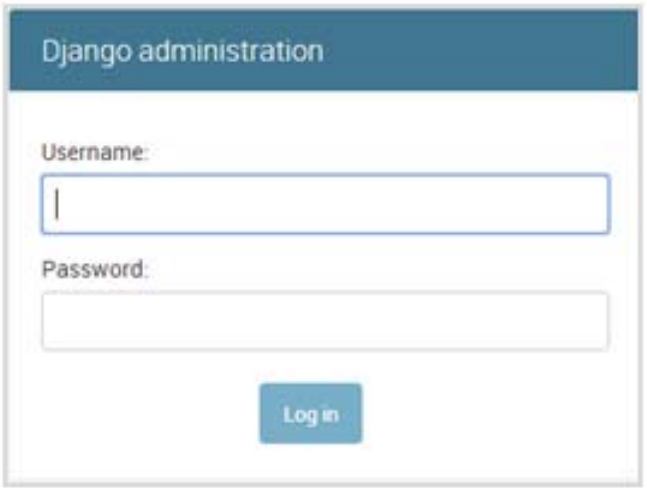

图 14.8　后台管理登录界面

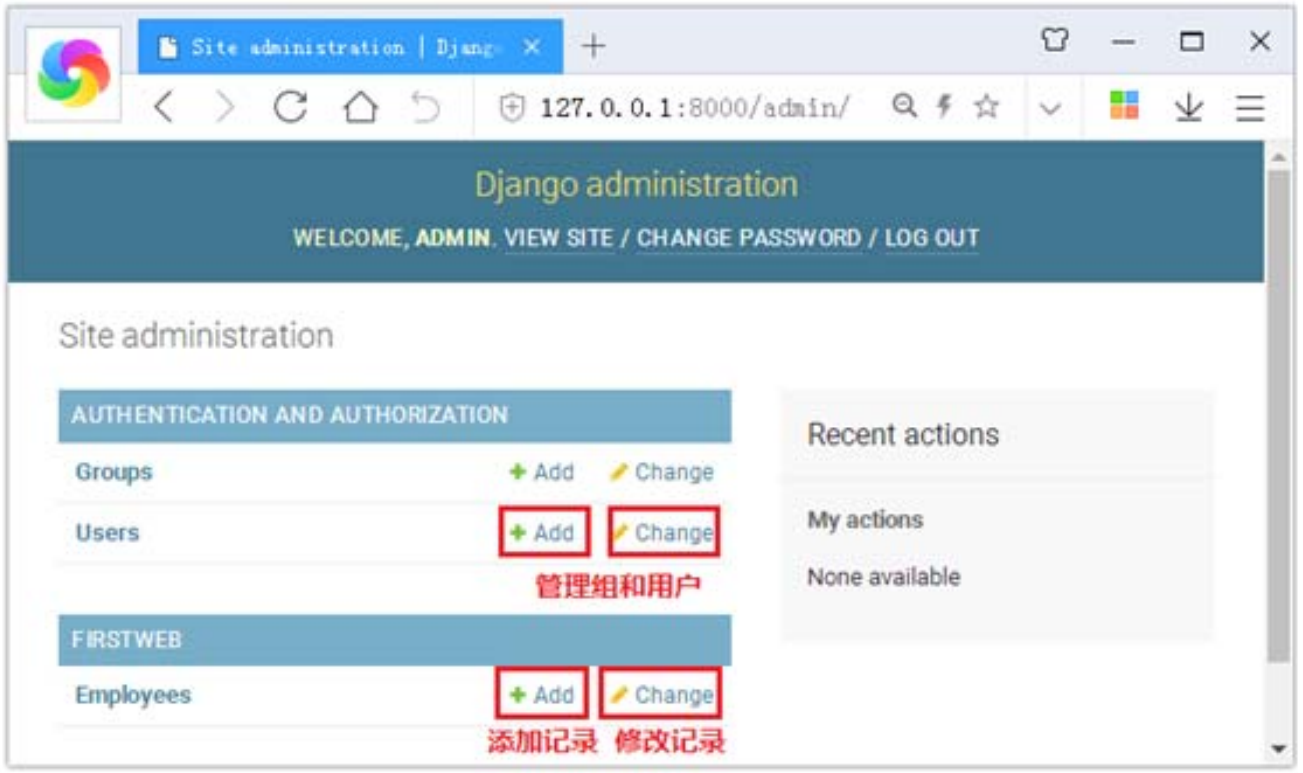

图 14.9　后台数据管理界面

14.4 典型案例

14.4.1 简单嗅探器

14.4.2 多线程端口扫描

14.4.3 用网络爬虫爬取全国城市天气信息

14.4.4 基于Django的个人博客

14.4.1 简单嗅探器

- ✓Sniffer(嗅探器)也叫抓取数据包软件，是一种监视网络数据运行的设备。
- ✓ Sniffer的工作原理是，利用以太网的特性把网络接口控制器(Network Interface Controller，NIC)置为混杂(Promiscuous)模式，使NIC能接收传输在网络上的每个信息包。
- ✓ 通过Sniffer，可以监视网络状态、数据流动情况，以及网络上传输的信息。
- ✓ Sniffer可通过软件或硬件实现。

14.4.1 简单嗅探器

【例14.7】 编程实现一个简单的Sniffer。

分析：使用Sniffer进行网络数据包抓取分析通常包括以下步骤。

(1) 创建套接字。

(2) 把网卡置于混杂模式。

(3) 捕获数据包。

(4) 分析数据包。

14.4.1 简单嗅探器

程序代码：

```
import socket, time

if __name__ == '__main__':
  ad = dict()
  LocalIP = socket.gethostbyname(socket.gethostname())
  socket1 = socket.socket(socket.AF_INET,socket.SOCK_RAW,socket.IPPROTO_IP)
  socket1.bind((LocalIP, 10000))
  socket1.setsockopt(socket.IPPROTO_IP,socket.IP_HDRINCL,1)
  socket1.ioctl(socket.SIO_RCVALL,socket.RCVALL_ON)
  start_time = time.clock()
```

14.4.1 简单嗅探器

```
#接收数据包.
  while True:
    pocket = socket1.recvfrom(65565)
    host_ip = pocket[1][0]
    if host_ip != LocalIP:                                    #过滤本机消息.
      ad[host_ip] = ad.get(host_ip,0) + 1
    end_time = time.clock()
    if end_time-start_time >= 30:                          #执行时间:30秒.
      break
  socket1.ioctl(socket.SIO_RCVALL, socket.RCVALL_OFF)      #关闭混杂模式.
  socket1.close()
  if ad.items():
    print("本机IP为 %s, 接收数据包情况如下:"%LocalIP)
  else:
    print("没有接收到数据包。请检查网络是否连通！ ")
  for item in ad.items():
    print("远程计算机IP:%s, 发包数目:%d."%(item[0],item[1]))
```

14.4.1 简单嗅探器

运行结果：

本机IP为 192.168.0.100, 接收数据包情况如下:

远程计算机IP:192.168.1.1, 发包数目:2.

远程计算机IP:14.116.136.253, 发包数目:1.

远程计算机IP:192.168.0.1, 发包数目:2.

远程计算机IP:180.163.237.176, 发包数目:1.

远程计算机IP:36.99.30.145, 发包数目:2.

远程计算机IP:36.110.237.201, 发包数目:1.

远程计算机IP:34.224.24.102, 发包数目:2.

14.4.2 多线程端口扫描

✓ 一个端口就是一个潜在的通信通道。
✓ 对目标计算机进行端口扫描，能得到许多有用的信息。
✓ 端口扫描的原理是，当一个主机向远端一个服务器的某个端口提出建立一个连接的请求时，如果对方有此项服务，就会应答；否则，对方仍无应答。
✓ 利用这个原理，如果对选定的某个范围的端口分别建立连接，并记录下远端服务器的应答，就可以知道目标服务器上安装了哪些服务，如对方是否提供**FPT**服务、**WWW**服务或其他服务。
✓ 常用服务及端口见表**14.5 (**教材**P272)** 。

【例14.8】 使用多线程和**Socket**测试远程主机端口开放情况。

分析：本案例实现方法如下。

(1)使用**Socket**连接，如果连接成功，则认为端口开放；否则，认为端口关闭。

(2)使用多线程**(**线程池**)**扫描指定远程主机的多个端口以加快扫描速度。

14.4.2 多线程端口扫描

程序代码：

```
import time,threadpool,socket

#扫描端口函数.
def scan_port(portList):
  global remote_host_ip
  try:
    soc = socket.socket(2,1)
    res = soc.connect_ex((remote_host_ip,portList))
    if res == 0:
      print('Port %s: OPEN'%portList)
      soc.close()
  except Exception as e:
    print("扫描端口异常:",e)
```

14.4.2 多线程端口扫描

```
if __name__ == '__main__':
  remote_host = input("请输入远程主机域名: ")
  remote_host_ip = ""
  try:
    remote_host_ip = socket.gethostbyname(remote_host)
    print('正在扫描远程主机 %s，请等候.' % remote_host_ip)
  except Exception as e:
    print("连接远程主机异常:%s."%repr(e))
  socket.setdefaulttimeout(1)        #超时时间(秒).
  portList = []
  for i in range(0,1024):
    portList.append(i)
```

14.4.2 多线程端口扫描

```
start_time = time.time()
  pool = threadpool.ThreadPool(10)              #创建线程池.
  reqs = threadpool.makeRequests(scan_port,portList)
  [pool.putRequest(req) for req in reqs]
  pool.wait()
  end_time = time.time()
  print("端口扫描时间: %d 秒."% (end_time-start_time))
```

14.4.2 多线程端口扫描

运行结果1(因特网连通):

请输入远程主机域名: **www.baidu.com**

正在扫描远程主机 **14.215.177.38**，请等候.

Port 80: OPEN

Port 443: OPEN

端口扫描时间: **51** 秒.

运行结果2(因特网未连通):

请输入远程主机域名: **www.baidu.com**

连接远程主机异常:**gaierror(11004, 'getaddrinfo failed').**

端口扫描时间: **0** 秒.

14.4.3 用网络爬虫爬取全国城市天气信息

✓ 网络爬虫(又被称为网页蜘蛛、网络机器人)是一种按照一定的规则自动地爬取万维网信息的程序或者脚本。

✓ **Python**爬虫架构主要由五个部分组成，分别是调度器、**URL**管理器、网页下载器、网页解析器和应用程序，见图**14.10**。

(1) 调度器

调度器主要负责调度**URL**管理器、网页下载器、网页解析器之间的协调工作。

(2) URL管理器

URL管理器包括待爬取的**URL**地址和已爬取的**URL**地址，防止重复和循环爬取**URL**。

14.4.3 用网络爬虫爬取全国城市天气信息

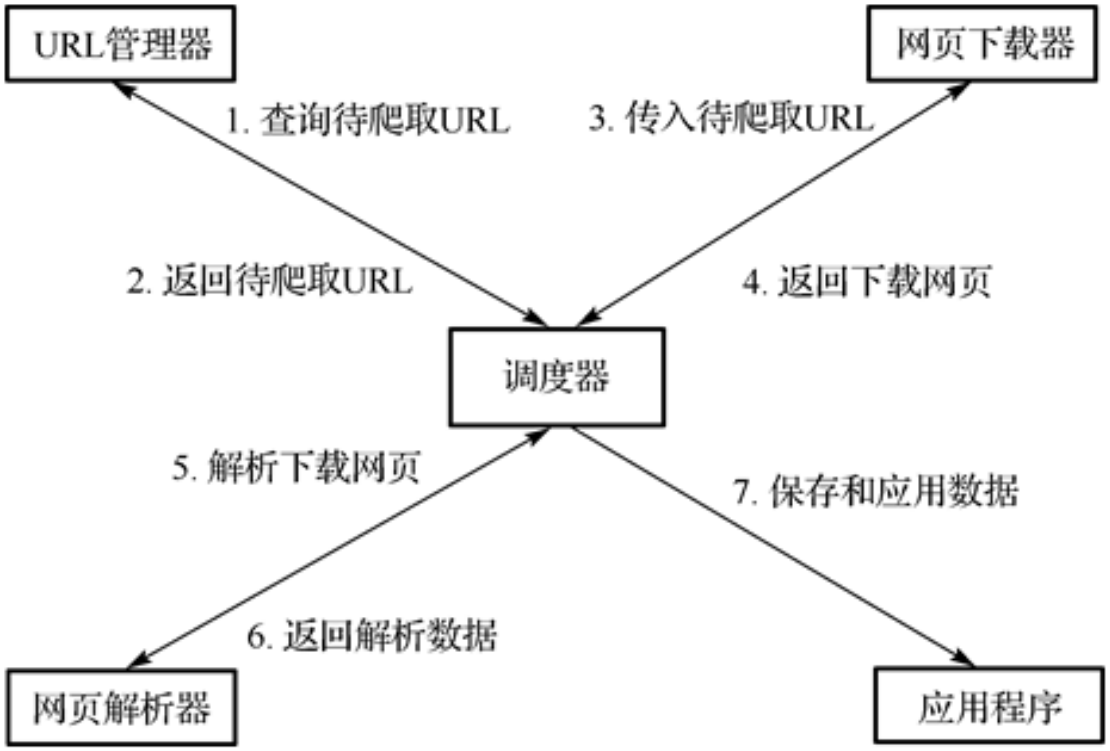

图 14.10　Python 爬虫的架构和流程

14.4.3 用网络爬虫爬取全国城市天气信息

(3) 网页下载器

✓ 网页下载器通过传入一个**URL**地址来下载网页，将网页转换成一个字符串。

✓ 用于实现下载网页的有**Cookie**、**Urllib(Python**自带模块**)**、**Requests(**第三方库**)**和**Scrapy**库等。

✓ 用得最广泛的是**Scrapy**库。

✓ **Scrapy**库是一个为爬取网站数据、提取结构性数据而编写的应用框架，可以应用在包括数据挖掘、信息处理或存储历史数据等一系列的程序中。

✓ **Scrapy**库的下载和安装参见**1.2.5**节。

(4) 网页解析器

✓ 网页解析器将一个网页字符串进行解析，按照要求提取有用的信息。

✓ 网页解析器可以使用正则表达式、**html.parser**、**beautifulsoup**、**lxml**等。

14.4.3 用网络爬虫爬取全国城市天气信息

(5) 数据保存和应用

✓ 可以将数据直接或处理后保存在数据库(如**SQLite**、**MySQL**或**MongoDB**)中；也可以将数据保存在一定格式的数据文件(如**CSV**文件)中。

✓ 可以直接应用在程序中；也可以进一步用于数据分析、挖掘等，探究其中的规律和内在关联。

14.4.3 用网络爬虫爬取全国城市天气信息

【例14.9】使用Scrapy库构建一个爬虫项目提取网站“www.weahter.com.cn”各个城市的天气信息并保存到SQLite3数据库中。

分析：

- ✓ 中国天气网(http://www.weather.com.cn/index.shtml)是中国气象局面向公众提供气象信息服务的核心门户
- ✓ 集成了中国气象局下属各业务部门最新的气象业务服务产品和及时丰富的气象资讯
- ✓ 是查询天气信息用得最多的一个网站。
- ✓ 通过网址http://www.weather.com.cn/省或直辖市的汉语拼音/index.shtml可以查询全国省份的各个城市或直辖市各个区的当日天气信息
- ✓ 通过网址http://www.weather.com.cn/weather/城市或区编号.shtml可以查询城市或区从当日算起的一周内的天气信息。

14.4.3 用网络爬虫爬取全国城市天气信息

- ✓ 其中，北京、上海、重庆、天津、香港、澳门、海南、吉林等省、市各个区的天气信息网页源代码和其他省、市有所不同，需要单独分析(详见源文件**CityWeather.py**中的代码)。
- ✓ 中国天气网一般每天在**7:30**、**11:30**、**18:00**等时间各发布一次全国主要城市天气信息。
- ✓ 在不同时间段发布的天气信息稍有差异，天气信息页面代码也稍有不同。
- ✓ 本案例只爬取中国天气网在每天**7:30**发布的天气信息。
- ✓ 读者可参考本案例方法，编写程序爬取在其他时间发布的各城市天气信息。

14.4.3 用网络爬虫爬取全国城市天气信息

1) 创建爬虫项目

✓ 进入存储爬虫项目代码的目录，运行如下命令创建爬虫项目**CityWeatherSpider**：

scrapy startproject CityWeatherSpider

✓ 进入**CityWeatherSpider**所在的项目文件夹，运行如下命令创建爬虫文件**CityWeather.py**：

scrapy genspider CityWeather.py www.weather.com.cn

上述命令运行成功后，将会生成如下结构的文件夹和文件。

(1) scrapy.cfg：项目的配置文件。

(2) CityWeatherSpider：项目的**Python**模块。

(3) CityWeatherSpider/items.py：项目的目标文件。

14.4.3 用网络爬虫爬取全国城市天气信息

(4) CityWeatherSpider/pipelines.py： 项目的管理文件。

(5) CityWeatherSpider/settings.py: 项目的设置文件。

(6) CityWeatherSpider/spiders/: 存储爬虫代码目录。

(7) CityWeatherSpider/spiders/CityWeather.py： 爬虫文件。

2) 分析天气网站源代码

✓ 以广东为例。

✓ 打开天气网站**http://www.weather.com.cn/guangdong/index.shtml**，找到天气信息所在位置(见图**14.11**)。

✓ 在网页上单击鼠标右键，选择“查看网页源代码”，找到与“城市预报列表”对应的位置(见图**14.12**)。

14.4.3 用网络爬虫爬取全国城市天气信息

图 14.11　网页-广东省各个城市的天气信息及链接

14.4.3 用网络爬虫爬取全国城市天气信息

```
<div class="forecast"><h1 class="weatheH1">城市预报列表
(2019-02-04 07:30发布)<span><img class="contraction" src="/m2/i/jian02.gif" /></span></h1>
<div class="forecastBox" id="forecastID">
<dl>
<dt>
<a title="广州天气预报" href="http://www.weather.com.cn/weather/101280101.shtml" target="_blank">广州</a>
</dt>
<dd>
<a href="http://www.weather.com.cn/static/html/legend.shtml" target="_blank"><img alt="" src="/m2/i/icon_weather/21x15/d01.gif" /><img alt="" src="/m2/i/icon_weather/21x15/n01.gif" /></a>
<a><span>27℃</span></a>/<a><b>17℃</b></a>
</dd>
</dl>
<dl>
<dt>
<a title="韶关天气预报" href="http://www.weather.com.cn/weather/101280201.shtml" target="_blank">韶关</a>
</dt>
```

图 14.12　网页源代码-广东省各个城市的天气信息及链接

14.4.3 用网络爬虫爬取全国城市天气信息

- ✓ 通过图14.12中源代码可获取广东省各个城市的天气信息及链接。
- ✓ 以广州市为例，打开广州市天气信息页面(见图14.13)。

```
<div class="crumbs fl">
    <a href="http://gd.weather.com.cn" target="_blank">广东</a>
    <span>></span>
    <a href="http://www.weather.com.cn/weather/101280101.shtml" target="_blank">广州</a><span>></span>  <span>城区</span>
</div>
```

图 14.13　广州市天气信息页面

14.4.3 用网络爬虫爬取全国城市天气信息

```
<ul class="t clearfix">
<li class="sky skyid lv3 on">
<h1>4日（今天）</h1>
<big class="png40 d01"></big>
<big class="png40 n01"></big>
<p title="多云" class="wea">多云</p>
<p class="tem">
<span>27</span>/<i>17℃</i>
</p>
<p class="win">
<em>
<span title="无持续风向" class="NNW"></span>
<span title="无持续风向" class="NNW"></span>
</em>
<i><3级</i>
</p>
```

图 14.14　广州市从当天算起一周内的天气信息

14.4.3 用网络爬虫爬取全国城市天气信息

3) 定义爬取信息

✓ 在源文件**items.py**中指定爬虫信息。

✓ 源文件**items.py**中的程序代码如下：

```
import scrapy
class CityweatherspiderItem(scrapy.Item):
  weather = scrapy.Field()
```

4) 编写爬虫代码

爬虫程序**CityWeather.py**完成如下功能。

(1) 获取各城市的天气信息网页。

(2) 从各城市的天气信息网页中获取该城市从当天算起一周内的天气信息。

(3) 将获取的天气信息通过**Item**返回。

程序代码参见教材**P276**。

14.4.3 用网络爬虫爬取全国城市天气信息

5) 指定爬虫数据处理方式

爬虫数据的处理通过文件**pipelines.py**完成，主要实现如下功能。

(1) 定义数据处理类**DataOpeation**。

类**DataOpeation**的功能是，创建或连接数据库、创建数据表、向数据表中添加或更新数据等。

(2) 调用数据处理类**DataOpeation**向数据表中添加或更新天气数据。

如果数据表没有某城市某天的天气数据，则将该城市某天的天气数据写入数据库；否则，更新该城市某天的天气数据。

程序代码参见教材**P278**。

14.4.3 用网络爬虫爬取全国城市天气信息

6) 指定数据处理程序

✓ 数据处理程序在源文件**settings.py**中指定。

✓ 源文件**settings.py**中的程序代码如下：

```
ITEM_PIPELINES = {
  'CityWeatherSpider.pipelines.CityweatherspiderPipeline':1,
}
```

7) 运行爬虫程序

✓ 进入爬虫项目所在目录**CityWeatherSpider**

✓ 运行如下命令启动爬虫程序：

```
scrapy crawl CityWeather
```

14.4.3 用网络爬虫爬取全国城市天气信息

运行结果有如下3种情况。

(1) 因特网连通且程序在指定时间范围(每天**7:30～11:30**)运行。程序将从“中国天气网”成功爬取各省(直辖市)的各城市(区)的天气信息，保存到数据库**WeatherDB.db**的数据表**table_weather**中(见图**14.15**)。

(2) 因特网未连通。程序运行后，将提示“无法连接服务器。请检查网络是否连通！”。

(3) 程序运行在指定时间范围外。程序运行后，将提示“请在**7:30～11:30**运行本程序**!!!**”，并关闭爬虫。

14.4.3 用网络爬虫爬取全国城市天气信息

ID	province	city	riqi	cloud	high	low	wind
1	北京	城区	2019-02-04	晴	4	6	东风<3级
2	北京	城区	2019-02-05	晴转多云	5	5	东南风<3级
3	北京	城区	2019-02-06	多云	4	5	北风<3级
4	北京	城区	2019-02-07	多云转阴	1	5	东南风<3级
5	北京	城区	2019-02-08	多云转晴	4	6	西北风3-4级转4-5级
6	北京	城区	2019-02-09	晴	1	9	西北风4-5级转<3级
7	北京	城区	2019-02-10	晴	0	8	西南风<3级
8	北京	延庆	2019-02-04	晴	1	11	东南风<3级
9	北京	延庆	2019-02-05	晴转多云	3	10	东南风<3级
10	北京	延庆	2019-02-06	多云	0	10	东南风<3级

图 14.15　保存到数据库中的北京天气信息

14.4.4 基于Django的个人博客

【例14.10】编程实现一个基于Django的简单个人博客。

分析如下。

1．个人博客功能

个人博客一般需要具有以下基本功能。

(1) 浏览博客列表。

(2) 查看博客详情，对博客进行评论。

(3) 添加新的博客、修改博客、删除博客等。

2．开发环境

开发环境包括Python 3.7.2、Django 2.2、PyCharm 2018.3.5、SQLite 3等。

3．实现步骤

1) 创建项目和应用

创建名为BlogWeb的项目和BlogApp的应用(创建方法参见【例14.3】)。

14.4.4 基于Django的个人博客

2) 模型设计和数据库迁移

(1) 模型设计。模型设计包括博客、博客类别、博客标签和评论。

① 博客类别**(Category)**。博客类别的属性包括：博客类别名称**(name)**。

② 博客标签**(Tag)**。博客标签的属性包括：博客标签名称**(name)**。

③ 博客**(Blog)**。博客的属性包括：博客标题**(title)**、作者**(author)**、内容**(content)**、创建时间**(create_time)**、修改时间**(modify_time)**、点击量**(click_nums)**、博客类别**(category)**和博客标签**(tag)**。

④ 评论**(Comment)**。评论的属性包括：博客**(blog)**、作者**(name)**、内容**(content)**和创建时间**(create_time)**。

(2) 数据库迁移。

✓ 执行命令**makemigrations**和**migrate**，生成名为**db.sqlite3**的数据库。

✓ **db.sqlite3**数据库包含以下**5**张表： **table_category**， **table_tag**， **table_blog**， **table_blog_tag**， **talbe_comment**。

14.4.4 基于Django的个人博客

3) 管理后台

① 创建超级用户。创建方法参见【例**14.6**】。

② 使用创建的超级用户登录并添加博客类别和标签。

4) 创建**form**

创建**form**在源文件**forms.py**中完成。

5) 设计视图

设计视图在源文件**views.py**中完成。

6) 配置**url**

配置**url**在源文件**urls.py**中完成。

7) 设计网页

博客中的网页包括博客列表**(bloglist.html)**、添加博客**(addblog.html)**、修改博客**(editblog.html)**和博客详情**(blogdetail.html)**等。

14.4.4 基于Django的个人博客

程序代码：参见教材**p281**。

运行结果：参见图**14.16**~图**14.19**。

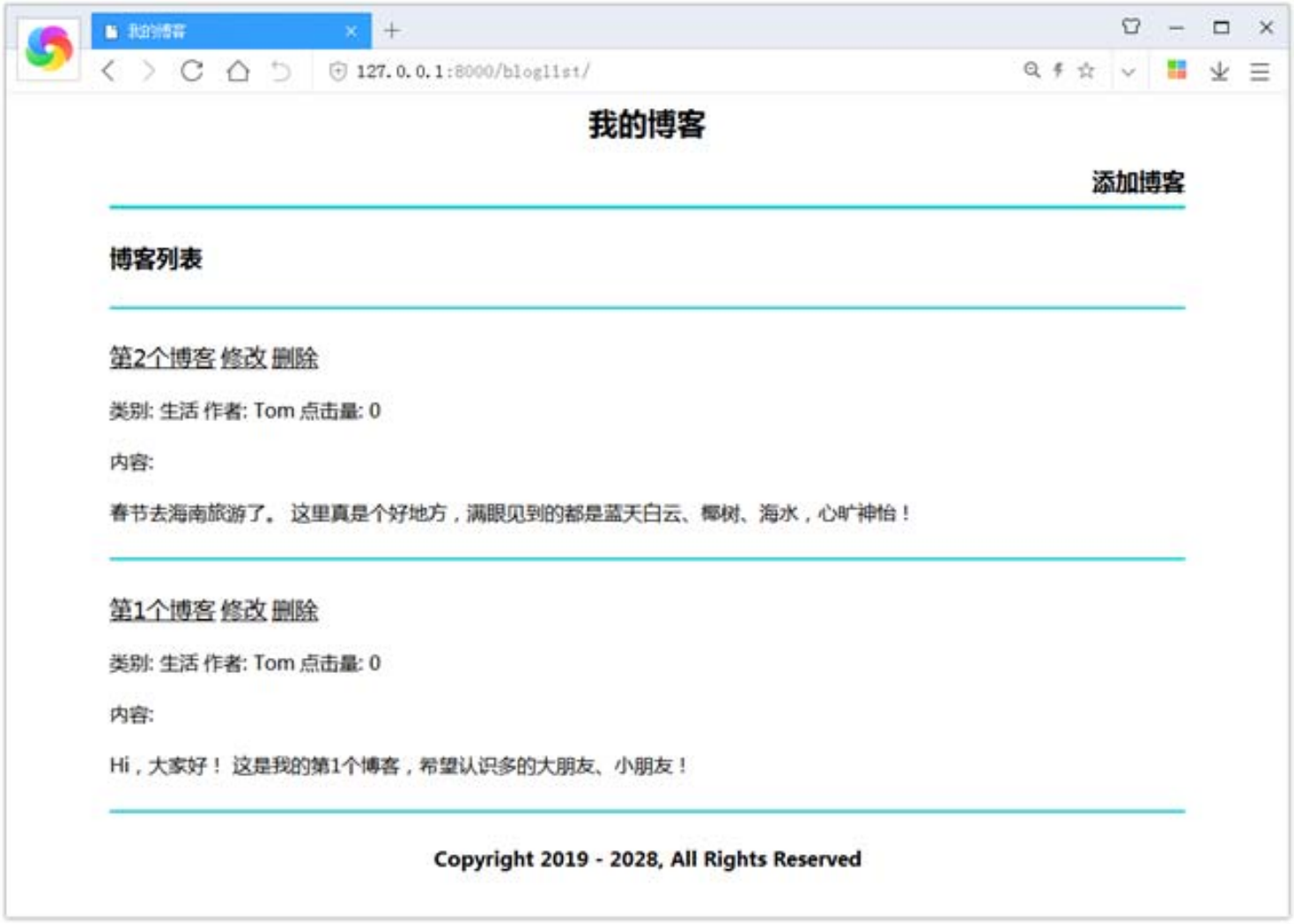

图 14.16　“博客列表”页面

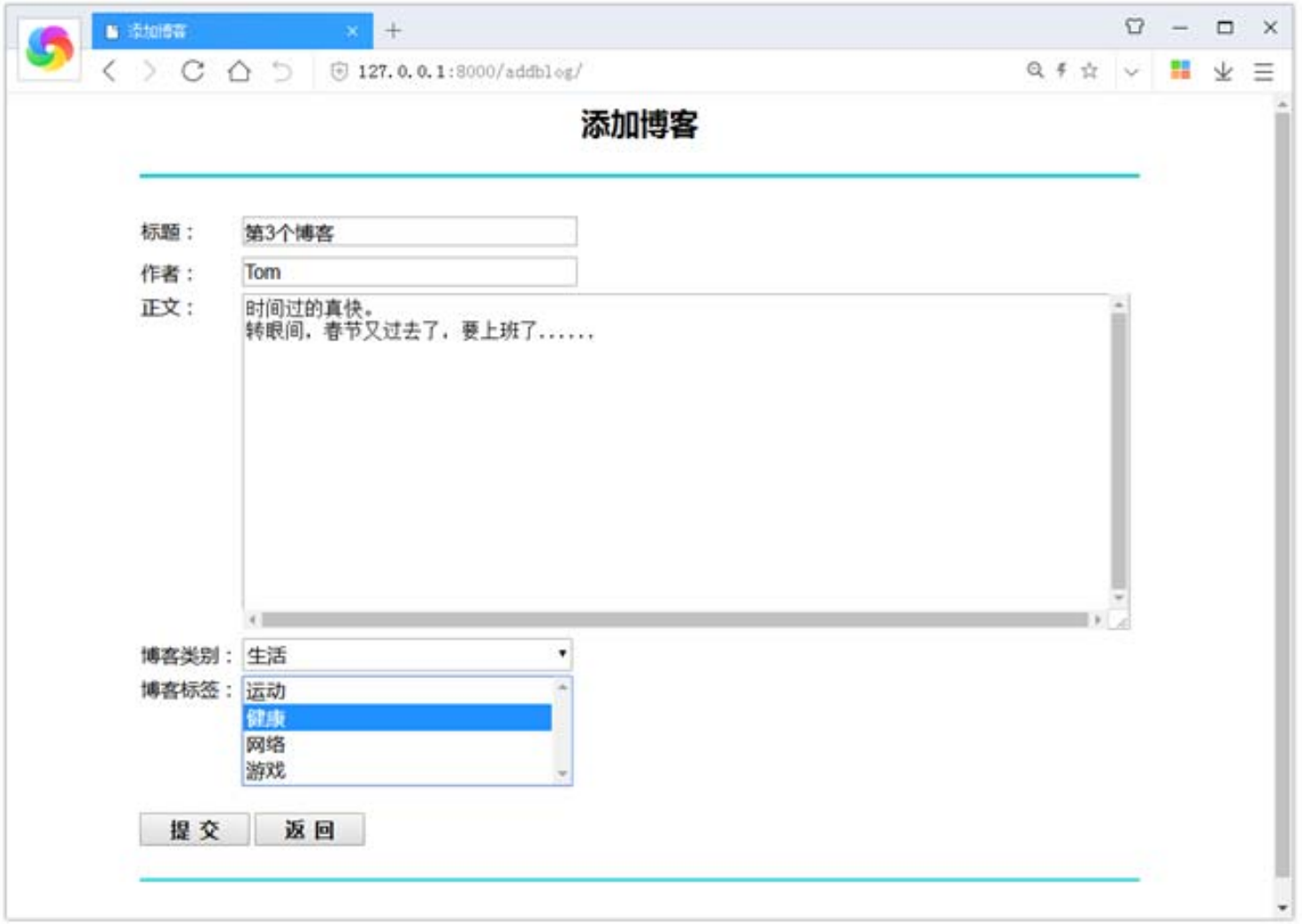

图 14.17　“添加博客”页面

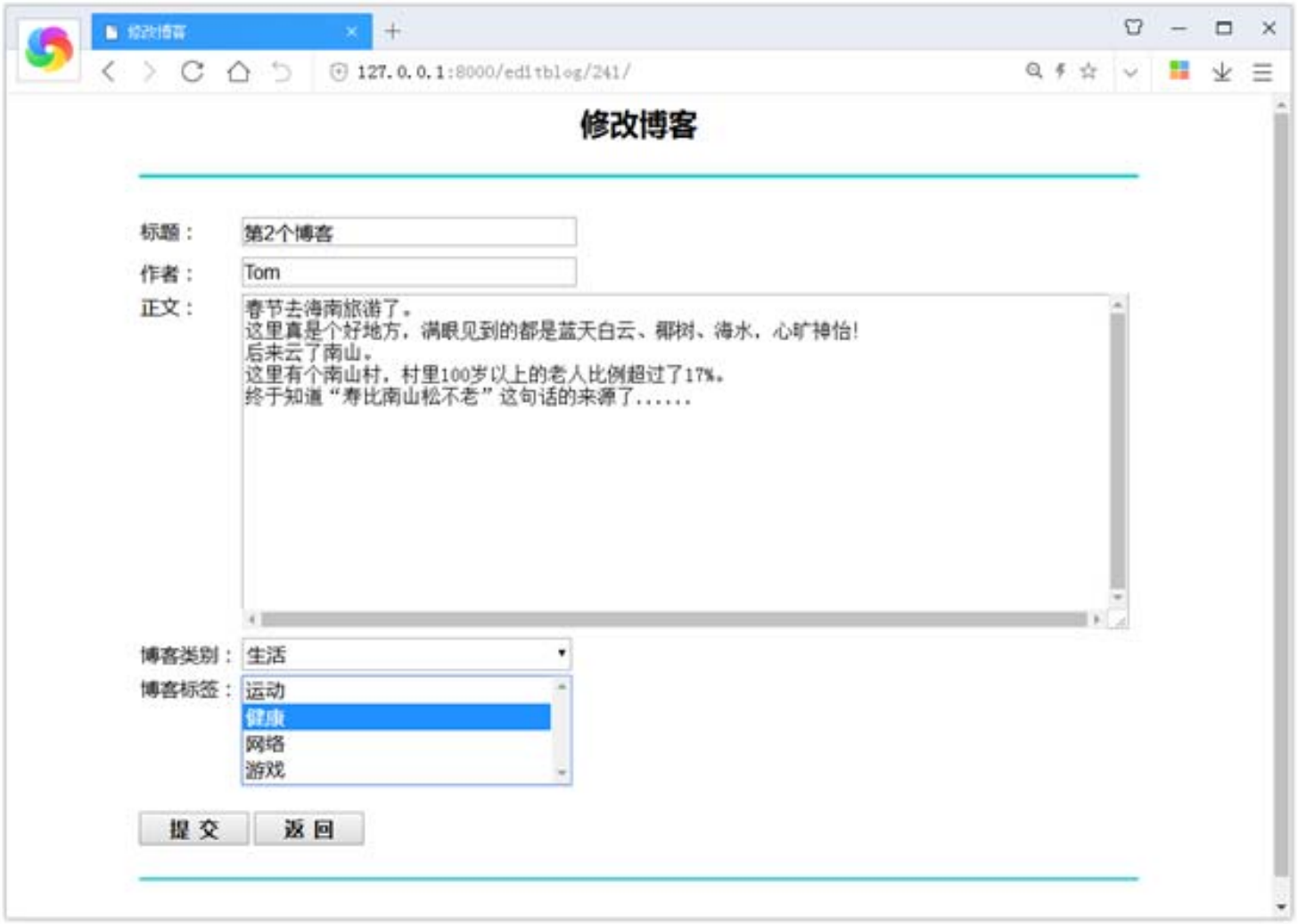

图 14.18　“修改博客”页面

图 14.19　“博客详情”页面

作业和实验

1. 练习题14 （P288）

作业要求：

- ✓ 简答题（在教材上以适当形式标出）
- ✓ 选择题和填空题（结果写在书上）

2. 实验14: 网络程序设计 （P313）

实验报告要求：

- ✓ 实验题目（文字）
- ✓ 源代码（文字）
- ✓ 运行结果（截图）