4.1 正则表达式与网络编程

正则表达式是一种强大的文本处理工具，用于匹配、搜索和替换字符串。网络编程允许不同计算机上的程序通过网络进行通信。本节将介绍正则表达式的基本概念和使用方法，以及网络编程的基础知识。

4.1.1 正则表达式的定义和使用

正则表达式是一种文本模式，包括普通字符（例如，字母a到z）和特殊字符（称为"元字符"）。它是一种在字符串中执行模式匹配以及"查找"和"替换"操作的技术。在Python中，正则表达式通过`re`模块提供支持。

定义正则表达式：

- **普通字符**：如`abc`将匹配字符串中的`abc`。
- **元字符**：具有特殊含义的字符，如：
  - `.`：匹配任意单个字符，除了换行符。
  - `^`：匹配字符串的开始。
  - `$`：匹配字符串的结束。
  - `*`：匹配0次或多次前面的字符。
  - `+`：匹配1次或多次前面的字符。
  - `?`：匹配0次或1次前面的字符。
  - `{n}`：匹配n次前面的字符。
  - `[...]`：匹配方括号内的任意单个字符。
  - `|`：匹配`|`前后的任意表达式。
  - `\`：转义特殊字符。

使用正则表达式：

1. 匹配：检查一个字符串是否匹配一个正则表达式。

```
import re
if re.match(r'\d+', '123abc'):
    print("Match found")
else:
    print("No match")
```

2. 搜索：在一个字符串中搜索匹配正则表达式的第一个位置。

```
match = re.search(r'\d+', 'abc123def')
if match:
    print("Found:", match.group())  # 输出找到的匹配
```

3. 查找所有匹配项：找到字符串中所有匹配正则表达式的部分。

```
matches = re.findall(r'\d+', 'abc123def456')
print("All matches:", matches)
```

4. 替换：替换字符串中所有匹配正则表达式的部分。

```
replaced = re.sub(r'\d+', '#', 'abc123def456')
print("Replaced string:", replaced)
```

5. 分割：使用正则表达式分割字符串。

```
parts = re.split(r'\d+', 'abc123def456ghi')
print("Parts:", parts)
```

正则表达式是处理文本数据的强大工具，能够简化复杂的字符串操作任务。在Python中，通过`re`模块的各种函数，可以方便地执行匹配、搜索、替换等操作。

4.1.2 正则表达式的使用

正则表达式的使用可以分为几个基本步骤：

1. 导入正则表达式库
对于不同的编程语言，首先需要导入或包含正则表达式的库。例如，在Python中，你需要导入`re`模块。

2. 定义正则表达式模式
根据需要匹配的文本，定义一个正则表达式模式。这个模式可以是简单的字符序列，也可以包含特殊字符和元字符来表示更复杂的模式。

3. 使用正则表达式函数
使用正则表达式库提供的函数，如匹配、搜索、替换或分割函数，来处理文本。

4. 处理匹配结果
处理函数返回的结果。这可能是一个布尔值（表示是否找到匹配项），也可能是匹配项本身，或者是匹配项的集合。

示例：Python中的正则表达式使用

以下是一个使用Python中`re`模块的简单示例，展示了如何查找字符串中的所有数字。

步骤一：导入`re`模块

```
import re
```

步骤二：定义正则表达式模式

```
pattern = r'\d+'  # 匹配一个或多个数字
```

步骤三：使用正则表达式搜索文本

```
text = "Example with 4 numbers, including 56 and 789."
matches = re.findall(pattern, text)
```

步骤四：处理匹配结果

```
print(matches)  # 输出：['4', '56', '789']
```

这个例子中，`findall`函数搜索文本中所有匹配正则表达式`pattern`的部分，并返回一个包含所有匹配项的列表。

4.1.3 常用的正则表达式模式

常用的正则表达式模式包括：

1. 数字：

- `\d`：匹配任何数字，等价于`[0-9]`。
- `\D`：匹配任何非数字字符，等价于`[^0-9]`。

2. 字母和数字：

- `\w`：匹配任何字母数字字符，等价于`[a-zA-Z0-9_]`。
- `\W`：匹配任何非字母数字字符，等价于`[^a-zA-Z0-9_]`。

3. 空白字符：

- `\s`：匹配任何空白字符，包括空格、制表符、换页符等等，等价于`[ \f\n\r\t\v]`。
- `\S`：匹配任何非空白字符，等价于`[^ \f\n\r\t\v]`。

4. 边界匹配：

- `^`：匹配输入字符串的开始位置。
- `$`：匹配输入字符串的结束位置。
- `\b`：匹配一个单词边界，即字与空格间的位置。
- `\B`：匹配非单词边界。

5. 字符类：

- `[abc]`：匹配任何包含括号内的字符（例如，`a`、`b`或`c`）。
- `[^abc]`：匹配任何不包含括号内的字符。
- `[a-z]`：匹配任何小写字母。
- `[A-Z]`：匹配任何大写字母。
- `[0-9]`：匹配任何数字。

6. 量词：

- `*`：匹配前面的子表达式零次或多次。

- `+`：匹配前面的子表达式一次或多次。
- `?`：匹配前面的子表达式零次或一次。
- `{n}`：匹配确定的`n`次。
- `{n,}`：至少匹配`n`次。
- `{n,m}`：最少匹配`n`次且不超过`m`次。

7. 特殊字符的转义：

- 使用`\`来转义特殊字符（例如，`\.`匹配点`.`字符本身，而不是任意字符）。

正则表达式中的元字符和模式可以组合使用，形成强大的匹配规则，用于各种文本处理任务，如数据验证、数据提取和数据替换等。以下是一些常用的正则表达式模式：

- **匹配数字**：`\d+`匹配一个或多个数字。
- **匹配字母**：`\w+`匹配一个或多个字母。
- **匹配空白字符**：`\s+`匹配一个或多个空白字符。
- **匹配任意字符**：`.`匹配任意单个字符。
- **匹配邮箱地址**：`\w+@\w+\.\w+`匹配邮箱地址。
- **匹配URL**：`https?://\w+\.\w+`匹配URL地址。
- **匹配日期**：`\d{4}-\d{2}-\d{2}`匹配日期格式。

4.1.4 网络编程基础和Socket编程

网络编程允许不同计算机上的程序通过网络进行通信。在网络编程中，套接字（Socket）是实现这种通信的基本构建块。以下是网络编程和套接字编程的基础知识：

网络编程基础

1. 网络协议：定义了数据通信的规则和格式。最常用的是TCP/IP协议族。

- **TCP**（传输控制协议）：提供可靠的、面向连接的通信。确保数据完整性和顺序。
- **UDP**（用户数据报协议）：提供无连接的通信。快速但不保证数据的完整性或顺序。

2. IP地址和端口号：网络中的每个设备都有一个唯一的IP地址，而端口号则用于标识设备上的特定服务。

3. DNS（域名系统）：将易于记忆的域名转换为IP地址。

4. 客户端-服务器模型：客户端发起请求，服务器响应请求。这是网络应用中最常见的架构模式。

Socket编程

套接字是网络通信的端点。使用套接字，程序可以指定网络层的协议（如TCP或UDP），并进行数据发送和接收。

- **创建套接字**：首先，需要创建一个套接字实例，指定使用的协议（TCP或UDP）。
- **绑定套接字**：服务器需要绑定套接字到一个IP地址和端口号，以便客户端可以连接。
- **监听连接**：服务器套接字监听来自客户端的连接请求。
- **接受连接**：服务器接受客户端的连接请求，建立连接。

- **数据交换**：一旦连接建立，客户端和服务器就可以开始数据交换。
- **关闭套接字**：通信结束后，双方需要关闭套接字以释放资源。

示例：Python中的简单TCP服务器和客户端

TCP服务器

```
import socket

# 创建套接字
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# 绑定到地址和端口
server_socket.bind(('localhost', 12345))
# 开始监听
server_socket.listen()

print("服务器启动，等待连接...")
# 接受连接
client_socket, address = server_socket.accept()
print(f"连接来自{address}")

# 接收数据
message = client_socket.recv(1024).decode('utf-8')
print(f"收到消息：{message}")
# 发送数据
client_socket.send("消息已收到".encode('utf-8'))

# 关闭套接字
client_socket.close()
server_socket.close()
```

TCP客户端

```
import socket

# 创建套接字
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# 连接到服务器
client_socket.connect(('localhost', 12345))

# 发送数据
client_socket.send("你好，服务器！ ".encode('utf-8'))
# 接收响应
response = client_socket.recv(1024).decode('utf-8')
print(f"服务器响应：{response}")

# 关闭套接字
client_socket.close()
```

这个例子展示了如何使用Python的`socket`库创建一个简单的TCP服务器和客户端。服务器监听本地主机的12345端口，客户端连接到服务器并发送消息，服务器接收消息并响应。

4.1.5 实践指导

正则表达式编程练习题

练习题1：提取所有的日期和时间

题目：编写一个Python脚本，使用正则表达式从给定的日志文件中提取所有的日期和时间。

正则表达式：

```
\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2}
```

Python代码：

```
import re

log_data = """
2023-04-01 12:00:00,INFO,User JohnDoe logged in from IP 192.168.1.1
2023-04-01 12:05:00,ERROR,Failed login attempt from IP 192.168.1.2
"""

pattern = r"\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2}"
matches = re.findall(pattern, log_data)
for match in matches:
    print(match)
```

练习题2：提取所有的IP地址

题目：编写一个Python脚本，使用正则表达式从给定的日志文件中提取所有的IP地址。

正则表达式：

```
\b(?:[0-9]{1,3}\.){3}[0-9]{1,3}\b
```

Python代码：

```
import re

log_data = """
2023-04-01 12:00:00,INFO,User JohnDoe logged in from IP 192.168.1.1
2023-04-01 12:05:00,ERROR,Failed login attempt from IP 192.168.1.2
"""

pattern = r"\b(?:[0-9]{1,3}\.){3}[0-9]{1,3}\b"
matches = re.findall(pattern, log_data)
for match in matches:
    print(match)
```

练习题3：提取所有的错误日志

题目：编写一个Python脚本，使用正则表达式从给定的日志文件中提取所有标记为ERROR的记录。

正则表达式：

```
^.*ERROR.*
```

Python代码：

```
import re

log_data = """
2023-04-01 12:00:00,INFO,User JohnDoe logged in from IP 192.168.1.1
2023-04-01 12:05:00,ERROR,Failed login attempt from IP 192.168.1.2
"""

pattern = r"^.*ERROR.*$"
matches = re.findall(pattern, log_data, re.MULTILINE)
for match in matches:
    print(match)
```

这些练习题和参考答案将帮助你理解如何使用正则表达式在Python中进行模式匹配和数据提取。

接下来请完成下面这个课程作业综合题。

课程作业：日志文件分析

给定一个服务器日志文件，日志格式如下：

```
2023-04-01 12:00:00,INFO,User JohnDoe logged in from IP 192.168.1.1
2023-04-01 12:05:00,ERROR,Failed login attempt from IP 192.168.1.2
2023-04-01 12:30:45,ERROR,Database connection failed
2023-04-01 13:00:00,INFO,User JaneDoe logged out
2023-04-01 13:15:00,INFO,User JohnDoe logged out
2023-04-01 13:20:00,ERROR,Failed login attempt from IP 192.168.1.3
2023-04-01 13:45:00,INFO,User MikeSmith logged in from IP 192.168.1.4
2023-04-01 14:00:00,INFO,User MikeSmith logged out
2023-04-01 14:30:00,INFO,User EmilyJones logged in from IP 192.168.1.5
2023-04-01 15:00:00,ERROR,Failed login attempt from IP 192.168.1.6
2023-04-01 15:30:00,INFO,User EmilyJones logged out
```

编写一个脚本来分析这个日志文件，实现以下功能：

1. 统计每种日志级别（INFO, ERROR等）的数量。
2. 找出所有尝试登录但失败的IP地址（假设登录失败会记录ERROR级别日志）。
3. 计算每个用户的在线时长（从登录到登出的时间差），假设每个用户在一天内只登录和登出一次。

Socket网络编程练习题

练习题1：文件传输服务器和客户端

题目描述：创建一个TCP服务器和客户端，客户端向服务器发送一个文件名请求，服务器根据请求返回文件内容。如果文件不存在，服务器返回一个错误消息。

服务器端代码：

```python
# File Transfer Server
import socket
import os

def file_transfer_server():
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind(('localhost', 8083))
    server_socket.listen(1)
    print("File transfer server is running on port 8083...")
    while True:
        client_socket, address = server_socket.accept()
        print(f"Connection from {address} has been established.")
        filename = client_socket.recv(1024).decode('utf-8')
        if os.path.exists(filename):
            with open(filename, 'rb') as file:
                client_socket.send(file.read())
        else:
            client_socket.send(b"Error: File not found.")
        client_socket.close()

file_transfer_server()
```

客户端代码：

```python
# File Transfer Client
import socket

def request_file(filename):
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client_socket.connect(('localhost', 8083))
    client_socket.send(bytes(filename, "utf-8"))
    response = client_socket.recv(4096)
    if response.startswith(b"Error:"):
        print(response.decode("utf-8"))
    else:
        with open(f"received_{filename}", 'wb') as file:
            file.write(response)
        print(f"File {filename} received successfully.")
    client_socket.close()

request_file("example.txt")
```

预期运行结果：

- 如果请求的文件存在，客户端将显示“File example.txt received successfully.”，并在当前目录下创建一个名为`received_example.txt`的文件。
- 如果文件不存在，客户端将显示“Error: File not found.”。

练习题2：聊天室服务器和客户端

题目描述：创建一个支持多客户端的TCP聊天室服务器。客户端连接到服务器后可以输入消息，服务器将消息广播给所有连接的客户端。

服务器端代码：

```python
# Chat Room Server
import socket
import threading

clients = []

def broadcast(message):
    for client in clients:
        client.send(message)

def handle_client(client):
    while True:
        try:
            message = client.recv(1024)
            broadcast(message)
        except:
            clients.remove(client)
            break

def chat_room_server():
```

```
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind(('localhost', 8084))
    server_socket.listen()
    print("Chat room server is running on port 8084...")
    while True:
        client, address = server_socket.accept()
        print(f"{address} connected.")
        clients.append(client)
        thread = threading.Thread(target=handle_client, args=(client,))
        thread.start()

chat_room_server()
```

客户端代码：

```
# Chat Room Client
import socket
import threading

def receive_message(client_socket):
    while True:
        try:
            message = client_socket.recv(1024)
            print(message.decode("utf-8"))
        except:
            print("An error occurred.")
            client_socket.close()
            break

def chat_room_client():
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client_socket.connect(('localhost', 8084))
    thread = threading.Thread(target=receive_message, args=(client_socket,))
    thread.start()
    while True:
        message = input('')
        client_socket.send(message.encode('utf-8'))

chat_room_client()
```

预期运行结果：

- 客户端连接到服务器后可以输入消息，服务器将消息广播给所有连接的客户端，实现简单的聊天室功能。

这些练习题和代码示例旨在提供实际应用场景中的Socket网络编程实践，包括文件传输和多客户端聊天室。
