

Chương 3

Lập trình mạng với các lớp InetAddress, URL và URLConnection

1. Lớp InetAddress

Các thiết bị được kết nối với mạng LAN có địa chỉ vật lý duy nhất. Điều này giúp cho các máy khác trên mạng trong việc truyền các gói tin đến đúng vị trí. Tuy nhiên, địa chỉ này chỉ có ích trong mạng LAN. Một máy không thể xác định được vị trí trên Internet bằng cách sử dụng các địa chỉ vật lý, vì các địa chỉ vật lý không chỉ ra vị trí của máy. Hơn nữa, các máy thường di chuyển từ vị trí này sang vị trí khác, trong trường hợp của máy xách tay hoặc máy palm chẳng hạn.

Những người lập trình mạng không cần phải quan tâm đến từng chi tiết dữ liệu được định tuyến như thế nào trong một mạng LAN. Hơn nữa, Java không cung cấp khả năng truy xuất tới các giao thức tầng liên kết dữ liệu mức thấp được sử dụng bởi LAN. Việc hỗ trợ như vậy là rất khó khăn. Vì mỗi kiểu giao thức sử dụng một kiểu địa chỉ khác nhau và có các đặc trưng khác nhau, chúng ta cần phải có các chương trình khác nhau cho mỗi kiểu giao thức mạng khác nhau. Thay vào đó, Java hỗ trợ giao thức TCP/IP, giao thức này có nhiều vụ liên kết các mạng với nhau.

Các thiết bị có một kết nối Internet trực tiếp được cung cấp một định danh duy nhất được gọi là địa chỉ IP. Các địa chỉ IP có thể là tĩnh hoặc động. Các địa chỉ IP được cấp phát động thường được sử dụng khi nhiều thiết bị cần truy cập Internet trong khoảng thời gian nhất định. Một địa chỉ IP chỉ có thể gắn với một máy, nó không thể dùng chung. Địa chỉ này được sử dụng bởi giao thức IP để định tuyến các datagram tới đúng vị trí. Không có địa chỉ, ta không thể liên lạc được với máy đó; vì thế tất cả các máy tính đều phải có một địa chỉ IP duy nhất.

Lớp `java.net.InetAddress` biểu diễn một địa chỉ Internet. Nó bao gồm hai trường thông tin: `hostname` (một đối tượng kiểu `String`) và `address` (một số kiểu `int`). Các trường này không phải là trường public, vì thế ta không thể truy xuất chúng trực tiếp. Lớp này được sử dụng bởi hầu hết các lớp mạng, bao gồm `Socket`, `ServerSocket`, `URL`, `DatagramSocket`, `DatagramPacket`,...

1.1. Tạo các đối tượng InetAddress

Lớp `InetAddress` được sử dụng để biểu diễn các địa chỉ IP trong một ứng dụng mạng sử dụng Java. Không giống với các lớp khác, không có các constructor cho lớp `InetAddress`. Tuy nhiên, lớp `InetAddress` có ba phương thức tĩnh trả về các đối tượng `InetAddress`

Các phương thức trong lớp `InetAddress`

- `public static InetAddress InetAddress.getByName(String hostname)`
- `public static InetAddress[] InetAddress.getAllByName(String hostname)`
- `public static InetAddress InetAddress.getLocalHost()`

Tất cả các phương thức này đều thực hiện kết nối tới server DNS cục bộ để biết được các thông tin trong đối tượng `InetAddress`

Ta xét phương thức đầu tiên. Phương thức này nhận tên của `hostname` làm tham số và trả về đối tượng kiểu `InetAddress`

Ví dụ:

```
try{
    InetAddress dc =InetAddress.getByName("www.microsoft.com");
    System.out.println(dc);
}
```

```

    }
    catch(UnknownHostException e)
    {
        System.err.println(e);
    }

```

Ví dụ 1:Viết chương trình nhận hostname từ đối dòng lệnh và in ra địa chỉ IP tương ứng với hostname đó.

```

import java.net.*;
public class TimDCIP
{
    public static void main(String[] args)
    {
        try{
            if(args.length!=1)
            {
                System.out.println("Cach su dung: java TimDCIP
<Hostname>");
            }
            InetAddress host = InetAddress.getBy_name(args[0]);
            String hostName = host.getHostName();
            System.out.println("Host name:"+hostName);
            System.out.println("Dia chi IP:"+host.getHostAddress());
        }
        catch(UnknownHostException e)
        {
            System.out.println("Khong tim thay dia chi");
            return;
        }
    }
}

```

1.2. Nhận các trường thông tin của một đối tượng InetAddress

Chỉ có các lớp trong gói java.net có quyền truy xuất tới các trường của lớp InetAddress. Các lớp trong gói này có thể đọc các trường của một đối tượng InetAddress bằng cách gọi phương thức getHostName và getAddress().

- public String getHostName(): Phương thức này trả về một chuỗi biểu diễn hostname của một đối tượng InetAddress. Nếu máy không có hostname, thì nó sẽ trả về địa chỉ IP của máy này dưới dạng một chuỗi ký tự.
- public byte[] getAddress() : Nếu bạn muốn biết địa chỉ IP của một máy, phương thức getAddress() trả về một địa chỉ IP dưới dạng một mảng các byte.

- Một số địa chỉ IP và một số mô hình địa chỉ có các ý nghĩa đặc biệt. Ví dụ, 127.0.0.1 là địa chỉ loopback. Các địa chỉ IPv4 trong khoảng 224.0.0.0 tới 239.255.255.255 là các địa chỉ multicast.

Java 1.5 thêm vào hai phương thức cho lớp `InetAddress` cho phép các ứng dụng kiểm tra liệu một nút cụ thể có đến được hay không với nút xuất phát là nút hiện hành; nghĩa là kiểm tra xem một liên kết mạng đã được thiết lập hay chưa. Các liên kết có thể bị phong tỏa vì nhiều nguyên nhân như firewall, các server ủy quyền, các router hoạt động sai chức năng, dây cáp bị đứt, hoặc host ở xa không bật.

- `public boolean isReachable(int timeout) throws IOException`
- `public boolean isReachable(NetworkInterface interface, int ttl, int timeout) throws IOException`

Các phương thức này cố gắng kết nối trên cổng echo trên host ở xa để tìm xem nó có thể đến được hay không. Nếu host đáp ứng trong khoảng thời gian timeout mili giây, các phương thức này trả về giá trị true nếu đến được, ngược lại nó trả về giá trị false.

1.3. Một số chương trình minh họa

Ví dụ 2 :Viết chương trình nhập một `hostName` từ đối dòng lệnh và in ra dòng thông báo cho biết địa chỉ IP tương ứng với địa chỉ IP đó thuộc lớp nào.

```
import java.net.*;
public class PhanLoaiDCIP
{
    public static void main(String[] args)
    {
        try{
            if(args.length!=1)
            {
                System.out.println("Cach su dung: java TimDCIP
<Hostname>");
            }
            InetAddress host = InetAddress.getByName(args[0]);
            String hostName = host.getHostName();
            System.out.println("Host name:"+hostName);
            System.out.println("Dia chi IP:"+host.getHostAddress());
            byte[] b=host.getAddress();
            int i=b[0]>=0?b[0]:256+b[0];

            if((i>=1)&(i<=126)) System.out.println(host+" thuoc dia chi lop A");
            if((i<=191)&(i>=128)) System.out.println(host+" thuoc dia chi lop
B");
            if((i<=223)&(i>=192)) System.out.println(host+" thuoc dia chi lop
C");

        }
        catch(UnknownHostException e)
```

```

        {
            System.out.println("Khong tim thay dia chi");
            return;
        }

    }
}

```

Trong chương trình này ta chỉ cần lưu ý dòng lệnh `int i=b[0]>=0?b[0]:256+b[0]`. Vì ta biết trong Java kiểu `byte` là kiểu số nguyên có dấu có khoảng giá trị là từ -128 đến 127. Do vậy, dòng lệnh `int i=b[0]>=0?b[0]:256+b[0]` làm nhiệm vụ chuyển đổi số nguyên có dấu ở dạng bù 2 về dạng số nguyên không dấu

2. Lớp URL

Cách đơn giản nhất để một chương trình Java định vị và tìm kiếm dữ liệu là sử dụng một đối tượng URL. Bạn không cần phải lo lắng tới các chi tiết bên trong của giao thức đang được sử dụng, khuôn dạng dữ liệu được nhận, hay làm thế nào để truyền tin với server; bạn chỉ cần cho biết URL, Java sẽ lấy dữ liệu về cho bạn.

Lớp `java.net.URL` là một khái niệm về bộ định vị tài nguyên thống nhất. Nếu lưu trữ URL dưới dạng một đối tượng String sẽ không có lợi so với việc tổ chức URL như một đối tượng với các trường : giao thức (protocol), hostname, cổng (port), đường dẫn (path), tên tập tin (filename), mục tài liệu (document section), mỗi trường có thể được thiết lập một cách độc lập.

2.1. Tạo các URL

Có bốn constructor khác nhau về thông tin mà nó cần. Constructor mà bạn sử dụng phụ thuộc vào thông tin mà bạn có, và khuôn dạng trong URL đó. Tất cả các constructor này sẽ đưa ra ngoại lệ `MalformedURLException` (URL không đúng khuôn dạng) nếu ta tạo ra một URL cho một giao thức mà nó không được hỗ trợ. URL cung cấp các hàm cấu tử sau:

- *`public URL(String url) throws MalformedURLException`*

Đây là constructor đơn giản nhất; tham số của nó chỉ là một URL ở dạng xâu.

Ví dụ

```

try{
    URL u = new URL("http://www.sun.com/index.html");
}
catch(MalformedURLException e)
{
    System.err.println(e);
}

```

- *`public URL(String protocol, String host, String file) throws MalformedURLException`*

Constructor này xây dựng một URL từ các xâu phân biệt xác định giao thức, hostname, và tệp tin. Port được thiết lập bằng -1 vì vậy cổng mặc định cho giao thức sẽ được sử dụng.

Ví dụ

```

try{
    URL u = new URL("http","/www.sun.com","index.html");
}
catch(MalformedURLException e){
    System.err.println(e);
}

```

- public URL(String protocol, String host, int port, String file) throws MalformedURLException

Trong một số ít trường hợp khi cổng mặc định không còn đúng, constructor này cho phép bạn xác định cổng một cách rõ ràng, là một số kiểu int. Các tham số khác giống như trên.

Ví dụ

```

try{
    URL u = new URL("http","/www.sun.com",80,"index.html");
}
catch(MalformedURLException e){
    System.err.println(e);
}

```

- public URL(URL u, String s) throws MalformedURLException

Hàm cấu tử này xây dựng một URL tuyệt đối từ URL tương đối; có thể đây là constructor bạn sẽ sử dụng thường xuyên.

Ví dụ

```
URL u1,u2;
```

```

try{
    URL u1= new URL("http://www.macfaq.com/index.html");
    URL u2 = new URL(u1,"vendor.html");
}
catch(MalformedURLException e)
{
    System.err.println(e);
}

```

Tên file được loại khỏi đường dẫn của u1, và tên file mới vendor.html được gán vào để tạo lên u2. Constructor này đặc biệt hữu ích khi bạn muốn duyệt qua một danh sách các file mà tất cả cùng nằm trong một thư mục.

2.2. Phân tích một URL thành các thành phần

Có bốn trường thông tin trong lớp URL: giao thức, port, file, mục tham chiếu tài liệu.

- public String getProtocol()
Phương thức getProtocol() trả về một chuỗi ký tự biểu diễn phần giao thức của URL
- public String getHost()

Phương thức `getHost()` trả về một chuỗi ký tự biểu diễn phần hostname của URL.

- `public int getPort()`

Phương thức `getPort()` trả về một số nguyên kiểu `int` biểu diễn số hiệu cổng có trong URL.

- `public int getDefaultPort()`

Phương thức `getDefaultPort()` trả về số hiệu cổng mặc định cho giao thức của URL

- `public String getFile()`

Phương thức `getFile()` trả về một chuỗi ký tự chứa phần đường dẫn của một URL; Java không phân chia một URL thành các phần đường dẫn và phần tệp tin riêng biệt.

- `public String getRef()`

Phương thức này trả về phần định danh đoạn của URL

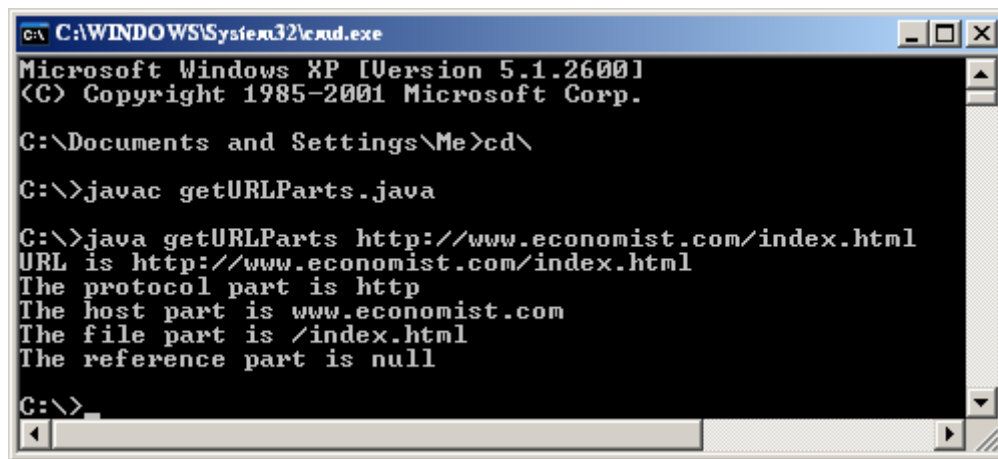
Ví dụ: Viết chương trình nhập vào một URL từ dòng lệnh và hiển thị từng thành phần tạo nên URL lên màn hình.

//Chương trình lấy thông tin của URL với các thông tin nhập từ bàn phím

```
import java.net.*;
```

```
class getURLParts{  
    public static void main(String[] args)  
    {  
        try  
        {  
            URL u = new URL(args[0]);  
            System.out.println("URL is "+u);  
            System.out.println("The protocol part is "+u.getProtocol());  
            System.out.println("The host part is "+u.getHost());  
            System.out.println("The file part is "+u.getFile());  
            System.out.println("The reference part is "+u.getRef());  
        }  
        catch(MalformedURLException e)  
        {  
            System.err.println(e);  
        }  
    }  
}
```

Kết quả thực hiện chương trình như sau:



```
C:\WINDOWS\System32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Me>cd\

C:\>javac getURLParts.java

C:\>java getURLParts http://www.economist.com/index.html
URL is http://www.economist.com/index.html
The protocol part is http
The host part is www.economist.com
The file part is /index.html
The reference part is null

C:\>
```

Hình 5.1

2.3. Tìm kiếm dữ liệu từ một URL

Nếu chỉ có URL thuần túy thì không có gì thú vị. Điều thú vị là dữ liệu nằm trong các tệp tin mà nó trỏ tới. Lớp `java.net.URL` có ba phương thức để tìm kiếm dữ liệu từ một URL

- *`public final InputStream openStream() throws java.io.IOException`*

Phương thức này kết nối tới một tài nguyên được tham chiếu bởi một URL, thực hiện cơ chế bắt tay cần thiết giữa client và server, và trả về một luồng nhập `InputStream`. Ta sử dụng luồng này để đọc dữ liệu. Dữ liệu nhận từ luồng này là dữ liệu thô của một tệp tin mà URL tham chiếu (mã ASCII nếu đọc một tệp văn bản, mã HTML nếu đọc một tài liệu HTML, một ảnh nhị phân nếu ta đọc một file ảnh). Nó không có các thông tin header và các thông tin có liên quan đến giao thức

- *`public URLConnection openConnection() throws java.io.IOException`*

Phương thức `openConnection()` mở một socket tới một URL xác định và trả về một đối tượng URL. Một đối tượng `URLConnection` biểu diễn một liên kết mở tới một tài nguyên mạng. Nếu lời gọi phương thức thất bại nó đưa ra ngoại lệ `IOException`.

- *`public final Object getContent() throws java.io.IOException`*

Phương thức này cung cấp cách thứ ba để tải dữ liệu được tham chiếu bởi một URL. Phương thức `getContent()` tìm kiếm dữ liệu được tham chiếu bởi một URL và chuyển nó thành một kiểu đối tượng nào đó. Nếu đối tượng tham chiếu tới một kiểu đối tượng văn bản nào đó như tệp tin ASCII hoặc tệp HTML, đối tượng được trả về thông thường sẽ là một kiểu luồng nhập `InputStream` nào đó. Nếu URL tham chiếu tới một đối tượng ảnh như ảnh GIF hoặc JPEG thì phương thức `getContent()` trả về đối tượng `java.awt.ImageProducer`

Ví dụ:Viết chương trình nhập một URL từ bàn phím, kết nối với Internet và hiển thị mã nguồn của trang Web đó lên màn hình.

```
import java.net.*;
import java.io.*;
public class ViewSource
{
    public static void main(String[] args)
    {
        URL u;
        String thisLine;
        if(args.length>0){
```

```

        try{
            u =new URL(args[0]);
            try{
                DataInputStream dis = new
DataInputStream(u.openStream());
                while((thisLine=dis.readLine())!=null)
System.out.println(thisLine);

            }
            catch(IOException e)
            {
                System.err.println(e);
            }
        }
        catch(MalformedURLException e){
            System.err.println(e);
        }
    }
}
}

```

Kết quả thực hiện chương trình

2.4. Các phương thức tiện ích

Lớp URL cung cấp hai phương thức tiện ích để thực hiện các thao tác trên URL. Phương thức `sameFile()` xác định xem hai URL có cùng trỏ tới một tài liệu hay không. Phương thức `toExternalForm()` chuyển đổi một đối tượng URL thành một xâu ký tự được sử dụng trong một liên kết HTML hoặc hộp thoại của trình duyệt.

- `public boolean sameFile(URL other)`

3. Lớp URLConnection

`URLConnection` là một lớp trừu tượng biểu diễn một liên kết tích cực tới một tài nguyên được xác định bởi một URL.

Lớp `URLConnection` có hai mục đích khác nhau nhưng liên quan với nhau.

- Thứ nhất, nó cung cấp nhiều khả năng điều khiển hơn thông qua việc tương tác với một server chứ không phải lớp URL. Với `URLConnection` ta có thể kiểm tra các header MIME được gửi bởi một Http Server phản ứng tương ứng. Ta cũng có thể sử dụng lớp `URLConnection` để tải về các tệp nhị phân. Ngoài ra `URLConnection` cho phép bạn gửi dữ liệu trở lại Web server bằng lệnh POST. Chúng ta sẽ xem tất cả các kỹ thuật đó trong chương này.
- Thứ hai, `URLConnection` là một phần của cơ chế quản trị giao thức, cơ chế này còn bao gồm cả lớp `URLStreamHandler`. Ý tưởng đằng sau các trình quản trị giao thức rất đơn giản: chúng cho phép bạn phân tách các chi tiết xử lý một giao thức với việc xử lý các kiểu dữ liệu cụ thể, cung cấp các giao diện người dùng, và thực

hiện các công việc khác mà một trình duyệt thường làm. Lớp cơ sở `URLConnection` là một lớp trừu tượng; để cài đặt một giao thức cụ thể bạn cần phải viết một lớp con. Các lớp con này có thể được tải bởi các ứng dụng của riêng bạn hay bởi các trình duyệt HotJava; trong tương lai, các ứng dụng Java có thể tải về các trình quản trị giao thức khi cần.

Mở các URLConnection

Một chương trình sử dụng lớp `URLConnection` trực tiếp theo một dãy các bước cơ bản sau:

- Xây dựng một đối tượng URL.
- Gọi phương thức `openConnection()` của đối tượng URL để tìm kiếm một đối tượng `URLConnection` cho URL đó.
- Cấu hình đối tượng URL.
- Đọc các trường header.
- Nhận một luồng nhập và đọc dữ liệu.
- Nhận một luồng xuất và ghi dữ liệu.
- Đóng liên kết.

Tuy nhiên, không phải lúc nào ta cũng phải thực hiện tất cả các bước này.

Ví dụ: Mở một `URLConnection` tới [http:// www.microsoft.com](http://www.microsoft.com)

```
import java.net.*;
import java.io.*;
public class getURLConnection
{
    public static void main(String[] args)
    {
        URL u;
        URLConnection uc;
        try
        {
            u= new URL("http://www.microsoft.com");
            try
            {
                uc=u.openConnection();
            }
            catch(IOException e)
            {
                System.err.println(e);
            }
        }
        catch(MalformedURLException e)
        {
            
```

```

        System.err.println(e);
    }
}
}

```

Mặc dù lớp `URLConnection` là một lớp trừu tượng nhưng nó có một phương thức được cài đặt. Phương thức đó là `connect()`; phương thức này tạo một liên kết tới một server; vì vậy nó phụ thuộc vào kiểu dịch vụ mà ta cài đặt (HTTP, FTP,...). Tất nhiên, ta có thể cảm thấy tiện lợi hay cần thiết phải nạp chồng các phương thức này trong lớp.

Rất nhiều các phương thức và các trường trong lớp `URLConnection` có là phương thức *protected*. Mặt khác ta chỉ có thể truy cập tới chúng thông qua các thể hiện của lớp `URLConnection` hoặc các lớp con của nó. Rất ít khi chúng ta khởi tạo và truy cập trực tiếp các đối tượng; thay vào đó, môi trường thời gian chạy sẽ tạo ra các đối tượng khi cần tùy thuộc vào giao thức sử dụng. Sau đó lớp sẽ được khởi tạo bằng cách sử dụng các phương thức `forName()` và `newInstance()` của lớp `java.lang.Class`.

- *public abstract void connect() throws IOException*

Phương thức `connect()` là một phương thức trừu tượng mở một liên kết tới một server. Tên của server được lấy ra từ một URL được lưu trữ như là một trường trong `URLConnection`, và được thiết lập bởi constructor của lớp. Các lớp con của lớp `URLConnection` nạp chồng các phương thức này để quản lý một kiểu liên kết cụ thể. Ví dụ, một phương thức `connect()` của lớp `FileURLConnection` chuyển đổi URL thành một filename trong thư mục tương ứng, tạo ra thông tin MIME cho file, và mở một luồng `FileInputStream` tới file. Phương thức `connect()` của lớp `HttpURLConnection` tạo ra một đối tượng `HttpClient` để kết nối với server. Phương thức `openConnection()` của đối tượng URL gọi phương thức `connect()` tương ứng, và trả về liên kết đã được mở. Vì vậy ta hiếm khi cần phải gọi phương thức `connect()` một cách trực tiếp.

Đọc dữ liệu từ một server

Dưới đây là các bước tối thiểu cần để tìm kiếm dữ liệu từ một URL bằng cách sử dụng đối tượng `URLConnection`:

- Bước 1: Xây dựng một đối tượng URL.
- Bước 2: Gọi phương thức `openConnection()` của lớp URL để tìm kiếm một đối tượng URL Connection cho đối tượng URL đó.
- Bước 3: Gọi phương thức `getInputStream()`.
- Bước 4: Đọc từ luồng nhập bằng cách sử dụng API.
- *public Object getContent() throws IOException*

Phương thức về mặt ảo giác giống như phương thức `getContent()` của lớp URL. Thực tế, phương thức `URL.getContent()` chỉ việc gọi phương thức `getContent()` tải về đối tượng được chọn bởi URL của `URLConnection` này. Để phương thức `getContent()` hoạt động, môi trường cần nhận dạng và hiểu kiểu nội dung. Hiện nay chỉ có một số kiểu nội dung được hiểu là `text/plain`, `image/gif`, và `image/jpeg`. Bạn có thể cài đặt thêm các kiểu trình quản lý nội dung khác có thể hiểu các kiểu nội dung khác. Phương thức `getContent()` chỉ làm việc với các giao thức như HTTP mà có một sự hiểu biết rõ ràng về các kiểu nội dung MIME. Nếu kiểu nội dung không được biết trước, hoặc giao thức không hiểu các kiểu nội dung, thì ngoại lệ `UnknownServiceException` được đưa ra.

- *public InputStream getInputStream()*

Phương thức `getContent()` chỉ làm việc khi Java có một trình quản lý nội dung cho kiểu nội dung. Nếu không phải trường hợp này, bạn có lẽ sẽ cần một luồng `InputStream` chung, luồng này cho phép bạn tự đọc và phân tích dữ liệu. Để làm

như vậy, cần gọi phương thức `getInputStream()`. Phương thức này cũng hữu ích khi trình quản lý nội dung có sẵn không thực hiện chính xác cái mà bạn muốn.

Ví dụ: Tải về một trang web thông qua một URL.

```
import java.net.*;
import java.io.*;

public class viewsource
{
    public static void main(String[] args)
    {
        String thisLine;
        URL u;
        URLConnection uc;
        if(args.length>0)
        {
            try{
                u =new URL(args[0]);
                try{
                    uc=u.openConnection();
                    DataInputStream theHtml = new
DataInputStream(uc.getInputStream());
                    try{

while((thisLine=theHtml.readLine())!=null)
                        {
                            System.out.println(thisLine);
                        }
                    }
                    catch(Exception e)
                    {
                        System.err.println(e);
                    }
                }
                catch(Exception e)
                {
                    System.err.println(e);
                }
            }
            catch(MalformedURLException e)
            {
                System.err.println(args[0]+" is not a parseable URL");
            }
        }
    }
}
```

```

        System.err.println(e);
    }

}

}

}

```

Phương thức `openStream()` của lớp `URL` trả về đối tượng `InputStream` từ đối tượng `URLConnection`.

- `public OutputStream getOutputStream()`

Đôi khi bạn cần phải ghi dữ liệu vào một `URLConnection`-chẳng hạn khi bạn muốn gửi dữ liệu tới một web server sử dụng lệnh `POST`. Phương thức `getOutputStream()` trả về một luồng `OutputStream` trên đó bạn có thể ghi dữ liệu để truyền tới một server. Vì một `URLConnection` không cho phép xuất kết quả ra ở chế độ mặc định, bạn phải gọi phương thức `setDoOutput()` trước khi yêu cầu một luồng xuất. Mỗi khi bạn có một luồng `OutputStream` thông thường bạn phải gắn nó với luồng `DataOutputStream` hoặc một lớp con khác của lớp `OutputStream` mà đưa ra nhiều đặc trưng hơn. Ví dụ:

```

try{
    URL u = new URL("http:// www.somehost.com/cgi-bin/acgi");
    URLConnection uc = u.openConnection();
    uc.setDoOutput(true);
    DataOutputStream dos = new DataOutputStream(uc.getOutputStream());
    dos.writeByte("Herre is some data");
}
catch(Exception e)
{
    System.err.println(e);
}

```

Sự khác biệt giữa `URL` và `URLConnection` là:

- `URLConnection` cho phép truy xuất tới header `HTTP`.
- `URLConnection` có thể cấu hình các tham số yêu cầu được gửi cho server.
- `URLConnection` có thể ghi dữ liệu lên server cũng như đọc dữ liệu từ server.

3.1. Phân tích Header

Multipurpose Internet Mail Extensions (MIME) là một tiêu chuẩn Internet mở rộng định dạng của email để hỗ trợ:

- Văn bản trong các bộ ký tự khác ASCII
- File đính kèm không phải văn bản
- Cơ quan thông điệp với nhiều phần
- Thông tin tiêu đề trong các bộ ký tự ASCII

HTTP Server cung cấp một số lượng thông tin đáng kể trong các header MIME trước mỗi đáp ứng. Thông tin trong các header MIME có thể bao gồm cơ chế mã hóa nội dung được sử dụng, ngày và giờ, chiều dài của nội dung được trả về bằng byte, ngày hết hạn của nội dung, ngày mà nội dung được sửa đổi lần cuối. Tuy nhiên, thông tin được gửi phụ thuộc vào server; một server nào đó gửi tất cả các thông tin này cho mỗi yêu cầu, các server khác gửi các thông tin nào đó, và một số ít server không gửi thông tin nào. Các phương thức của mục này cho phép ta truy vấn một `URLConnection` để tìm ra thông tin MIME nào mà server đã cung cấp.

Ngoài HTTP, rất ít giao thức sử dụng các header MIME. Khi viết lớp con của lớp `URLConnection`, thông thường cần phải nạp chồng các phương thức này sao cho chúng trả về các giá trị có ý nghĩa. Phần thông tin quan trọng nhất bạn có thể thiếu là kiểu nội dung MIME. `URLConnection` cung cấp một số phương thức tiện ích nào đó mà trợ giúp bạn đoán nhận ra kiểu nội dung, dựa trên tên file của nó hoặc một số byte đầu tiên của chính dữ liệu.

- `public String getContentType()`

Phương thức trả về kiểu nội dung MIME của dữ liệu. Nó phụ thuộc vào web server gửi một header MIME tương ứng, bao gồm một kiểu nội dung xác thực. Nó không đưa ra ngoại lệ và trả về giá trị null nếu kiểu nội dung không có. `text/html` sẽ là kiểu nội dung mà bạn thường xuyên gặp nhất khi kết nối với web server. Các kiểu nội dung phổ biến khác bao gồm: `text/plain`, `image/gif`, `image/jpeg`.

- `public int getContentLength()`

Phương thức này cho ta biết nội dung có kích thước bao nhiêu byte. Rất nhiều server chỉ gửi các header độ dài nội dung khi chúng truyền một file nhị phân, chứ không phải khi truyền một file văn bản. Nếu không có chiều dài nội dung, phương thức `getContentLength()` trả về -1. Phương thức này không đưa ra ngoại lệ. Nó được sử dụng khi ta cần biết cần đọc bao nhiêu byte, hoặc khi ta cần tạo ra một buffer đủ lớn để lưu trữ dữ liệu.

- `public String getContentEncoding()`

Phương thức này trả về String cho ta biết cách thức mã hóa. Nếu nội dung được gửi không được mã hóa (như trong trường hợp của HTTP server), phương thức này trả về giá trị null. Nó không đưa ra ngoại lệ.

- `public long getDate()`

Phương thức `getDate()` trả về một số nguyên kiểu long cho bạn biết tài liệu đã được gửi khi nào. Ta có thể chuyển đổi nó sang một đối tượng kiểu `java.util.Date`. Ví dụ:

```
Date docSent = new Date(uc.getDate());
```

Đây là thời gian tài liệu được gửi trên server. Nếu header MIME không có một header `Date`.

- `public long getExpiration()`

- `public long getLastModified()`

Phương thức `date`, `getLastModified()`, trả về ngày mà tài liệu được sửa đổi lần cuối

Ví dụ: Đọc các URL từ dòng lệnh, và sử dụng 6 phương thức để in ra kiểu nội dung, chiều dài nội dung, mã hóa nội dung, ngày sửa đổi cuối cùng, ngày hết hạn, và ngày hiện hành.

3.2. Tìm kiếm các trường Header MIME

Sáu phương thức cuối cùng đòi hỏi các trường nhất định từ header MIME, nhưng không có giới hạn nào về số các trường header mà một thông điệp MIME có thể có. Năm phương thức tiếp theo kiểm tra các trường nhất định trong header MIME.

`URLConnection` không có các header MIME thực sự, vì vậy tất cả các phương thức này trả về giá trị null khi bạn đang làm việc với một file: URL, hành vi mặc định của chúng. `URLConnections` tìm ra một trường header để thỏa mãn yêu cầu của bạn. Nếu được tìm thấy, nó được trả về, ngược lại nó trả về giá trị null.

- `public String getHeaderField(String name)`

Phương thức `getHeaderField()` trả về giá trị của trường header MIME được đặt tên. Tên của header không phân biệt chữ hoa và chữ thường và không chứa dấu kết thúc.

Ví dụ, để tìm giá trị của các trường header `Content-type`, `Content-encoding` của một đối tượng `URLConnection uc` bạn có thể viết:

```
uc.getHeaderField("content-type");
uc.getHeaderField("content-encoding");
```

Để nhận thông tin về các trường `Date`, `Content-length`, hoặc `Expiration` bạn cũng thực hiện tương tự:

```
uc.getHeaderField("date");
uc.getHeaderField("expires");
uc.getHeaderField("Content-length");
```

Tất cả các phương thức này đều trả về các `String`, không phải `int` cũng không phải `long` như các phương thức `getContentLength()`, `getExpirationDate()`, `getLastModified()`, và `getDate()`. Nếu bạn quan tâm đến một giá trị số, bạn phải chuyển đổi `String` thành `long` hoặc `int`.

- `public String getHeaderFieldKey(int n)`

Phương thức này trả về khóa (nghĩa là tên trường: ví dụ, `Content-length` hoặc `Server`) của trường header thứ `n`. Header đầu tiên là 0. Ví dụ, để nhận khóa thứ 6 của header MIME của `URLConnection`, bạn viết:

```
String header5=uc.getHeaderFieldKey(5);
```

- `public String getHeaderField(int n)`

Phương thức này trả về giá trị trường header MIME thứ `n`. Header MIME đầu tiên là một.

Ví dụ: Sử dụng phương thức kết hợp với phương thức `getHeaderFieldKey()` để in ra header MIME.

- `public long getHeaderFieldDate(String name, long default)`

Phương thức này trước hết tìm kiếm trường header được xác định bởi tham số `name` và cố gắng chuyển đổi xâu này sang kiểu `long`.

3.3. Các phương thức `RequestProperty`

Bốn phương thức sau không thực hiện bất kỳ công việc gì trong lớp cơ sở `URLConnection`, cũng không được cài đặt trong các lớp `URLConnection` hoặc `HttpConnection`. Bạn có thể mong muốn nạp chồng chúng trong một lớp con để cài đặt phương thức tra cứu bằng băm, chẳng hạn để xây dựng một bảng băm chứa tất cả các header MIME của yêu cầu.

- `public String getRequestProperty(String property_name)`

Phương thức này đưa ra một ngoại lệ `IllegalAccessError` nếu liên kết là mở, ngược lại phương thức trả về giá trị `null`. Nếu bạn nạp chồng nó, các phương thức của bạn cần trả về giá trị gắn với một thuộc tính cho trước như là một xâu.

- `public static void setDefaultRequestProperty(String property_name, String property_value)`

Phương thức này không thực hiện công việc gì. Nếu bạn nạp chồng phương thức này, bạn sẽ sử dụng nó để lưu trữ một giá trị mặc định cho thuộc tính cho trước.

- `public void setRequestProperty(String property_name, String property_value)`

Phương thức này trả về ngoại lệ `IllegalAccessError` nếu liên kết đang mở. Ngược lại nó không thực hiện gì. Nếu bạn nạp chồng nó, bạn sẽ sử dụng nó để lưu trữ giá trị của một thuộc tính cho trước.

- `public String getDefaultRequest(String property_name)`

Phương thức này luôn trả về giá trị `null`. Nếu bạn nạp chồng phương thức này, bạn cần trả về giá trị mặc định được gán cho một thuộc tính cho trước như là một `String`.

- `protected URLConnection(URL u)`

Constructor trong `URLConnection` nhận một tham số là `URL` để thực hiện việc liên kết. Tất cả các tính chất khác của một `URLConnection` ban đầu được thiết lập là các giá trị mặc định của chúng và bạn có thể thay đổi chúng bằng tập các phương thức. Vì constructor có tính chất `protected`, chỉ có các đối tượng trong gói `java.net` mới có thể tạo ra một `URLConnection`. `URLConnection` là một lớp trừu tượng vì vậy bạn chỉ có thể gọi constructor của nó từ constructor của một trong các lớp con của nó.

Nếu bạn đang tạo ra lớp con của lớp `URLConnection`, bạn phải gọi constructor này trong dòng đầu của constructor của lớp con như sau:

```
myURLConnection(URL u)
{
    super(u);
}
```

Nếu bạn không đưa vào một lời gọi cụ thể tới constructor trong lớp con của bạn, Java cố gắng tạo ra một constructor không tham số của lớp cha: ví dụ `URLConnection()`. Vì lớp `URLConnection` không cung cấp các constructor không tham số, loại bỏ lời gọi cụ thể sẽ gây ra lỗi biên dịch.

3.4. Các trường và các phương thức có liên quan

Có mười ba trường trong lớp `java.net. URLConnection`. Bảy trường là các biến tĩnh định nghĩa các giá trị mặc định cho các thể hiện của lớp `URLConnection`. Sáu phương thức khác định nghĩa trạng thái của một đối tượng `URLConnection` cụ thể. Một vài phương thức `get` và `set` thay đổi các giá trị của chúng.

Hầu hết các phương thức thiết lập các trường đưa ra ngoại lệ `IllegalAccessExceptions` nếu bạn gọi chúng trong khi liên kết đang mở. Nhìn chung, bạn chỉ có thể thiết lập các thuộc tính của một đối tượng `URLConnection` trong khi liên kết đóng.

- `protected URL url`
- `public URL getURL()`

Trường url xác định URL mà URLConnection liên kết tới nó. Nó được thiết lập bởi constructor khi bạn tạo ra một URLConnection, và không cần thay đổi. Bạn có thể tìm kiếm giá trị bằng cách gọi phương thức `getURL()`.

```
import java.net.*;
import java.io.IOException;
public class printURLConnection
{
    public static void main(String[] args)
    {
        URL u;
        URLConnection uc;
        try{
            u = new URL("http://www.ora.com/");
            try
            {
                uc=u.openConnection();
                System.out.println(uc.getURL());
            }
            catch(IOException e)
            {
                System.err.println(e);
            }
        }
        catch(MalformedURLException e)
        {
            System.err.println(e);
        }
    }
}
```

- protected boolean connected

Trường `connected` là đúng nếu liên kết là mở và là sai nếu liên kết đóng. Vì bạn không mở liên kết khi một đối tượng URLConnection được tạo ra, giá trị ban đầu của nó là `false`. Bạn chỉ có thể truy cập tới biến này thông qua các thể hiện của lớp URLConnection và các lớp con của nó. Không có các phương thức đọc hoặc thay đổi giá trị của nó. Khi viết một trình quản trị giao thức, bạn có nhiệm vụ thiết lập giá trị của biến này là `true` và thiết lập lại nó bằng giá trị `false` khi liên kết đóng. Rất nhiều phương thức trong gói `java.net`. URLConnection đọc biến này để xác định xem chúng có thể thực hiện việc gì. Nếu việc thiết lập không chính xác chương trình của bạn sẽ gặp các lỗi không dễ chuẩn đoán.

- protected boolean allowUserInteraction

- `public void setAllowUserInteraction(boolean allowuserinteraction)`
- `public boolean getAllowUserInteraction()`

Một số `URLConnection` cần tương tác với người dùng. Ví dụ, một trình duyệt web có thể yêu cầu username và password. Tuy nhiên, rất nhiều ứng dụng không thể khẳng định một người sử dụng đang có mặt để tương tác với nó. Như ý nghĩa của tên, `allowUserInteraction` xác định xem liệu tương tác người sử dụng có được phép hay không. Ở chế độ mặc định nó được thiết lập là `false`. Vì biến này là `protected`, ta có thể sử dụng phương thức `getAllowUserInteraction()` để đọc giá trị của nó và sử dụng phương thức `setAllowUserInteraction()` để thiết lập giá trị của nó. Giá trị `true` chỉ ra rằng tương tác với người sử dụng là được phép; giá trị `false` chỉ ra rằng không có tương tác với người dùng. Giá trị có thể được đọc ở bất kỳ thời điểm nào, nhưng nó chỉ có thể được thiết lập khi liên kết bị đóng. Gọi phương thức `setAllowUserInteraction()` khi liên kết mở sẽ đưa ra ngoại lệ `IllegalAccessError` (chứ không phải là `IllegalAccessException`). Các chương trình thường không đón bắt các lỗi (không giống như các ngoại lệ); một lỗi không được đón bắt thường buộc chương trình phải kết thúc.