Nguyễn Đức Nam :18103200029

Chu Phú Tùng :18103200031

Vũ Duy Công :18103200005

Môn:Thực tập lập trình mạng

Thầy giáo:Trần Hồng Việt

[16.1 What Is a Protocol Handler? Xử lý giao thức là gì? 2](#_Toc73609646)

[16.2 The URLStreamHandler Class 9](#_Toc73609647)

[16.2.1 The Constructor 9](#_Toc73609648)

[16.2.2 Methods for Parsing URLs - Phương thức chuyển URL 10](#_Toc73609649)

[16.2.2.1 Protected void parseURL(URL u, String spec, int start, int limit) 10](#_Toc73609650)

[16.2.2.2 Protected String toExternalForm(URL u) 13](#_Toc73609651)

[16.2.2.3 Protected void setURL(URL u, String protocol, String host, int port, String file, String ref ) 14](#_Toc73609652)

[16.2.2.4 Protected int getDefaultPort( ) // Java 1.3 17](#_Toc73609653)

[16.2.2.5 Protected InetAddress getHostAddress(URL u) // Java 1.3 17](#_Toc73609654)

[16.2.2.6 Protected boolean hostsEqual(URL u1, URL u2) // Java 1.3 18](#_Toc73609655)

[16.2.2.7 Protected boolean sameFile(URL u1, URL u2) // Java 1.3 19](#_Toc73609656)

[16.2.2.8 Protected boolean equals(URL u1, URL u2) // Java 1.3 19](#_Toc73609657)

[16.2.3 A Method for Connecting 20](#_Toc73609658)

[16.3 Writing a Protocol Handler 23](#_Toc73609659)

[16.4 More Protocol Handler Examples and Techniques - Ví dụ và kỹ cách kỹ thuật xử lý giao thức 30](#_Toc73609660)

[16.4.1 A daytime Protocol Handler - Trình xử lý giao thức ban ngày 32](#_Toc73609661)

[16.4.2 A chargen Protocol Handler - Trình xử lý giao thức hàng loạt 37](#_Toc73609662)

[16.5 The URLStreamHandlerFactory Interface 44](#_Toc73609663)

Protocol Handlers – Xử lý giao thức

When designing an architecture that would allow them to build a self-extensible browser, the engineers at Sun divided the problem into two parts: handling protocols and handling content. Handling a protocol means taking care of the interaction between a client and a server: generating requests in the correct format, interpreting the headers that come back with the data, acknowledging that the data has been received, etc. Handling the content means converting the raw data into a format Java understands, for example, an InputStream or an AudioClip. These two problems, handling protocols and handling content, are distinct. The software that displays a GIF image doesn't care whether the image was retrieved via FTP, HTTP, gopher, or some new protocol. Likewise, the protocol handler, which manages the connection and interacts with the server, doesn't care if it's receiving an HTML file or an MPEG movie file; at most, it will extract a content type from the headers to pass along to the content handler.

Java divides the task of handling protocols into a number of pieces. As a result, there is no single class called ProtocolHandler. Instead, pieces of the protocol handler mechanism are implemented by four different class es in the java.net package: URL, URLStreamHandler, URLConnection, and URLStreamHandlerFactory. URL is the only concrete class in this group; URLStreamHandler and URLConnection are both abstract classes, and URLStreamHandlerFactory is an interface. Therefore, if you are going to implement a new protocol handler, you have to write concrete subclasses for the URLStreamHandler and the URLConnection. To use these classes, you may also have to write a class that implements the URLStreamHandlerFactory interface.

Khi thiết kế một kiến ​​trúc cho phép họ xây dựng một trình duyệt có thể tự mở rộng, các kỹ sư tại Sun đã chia vấn đề thành hai phần: xử lý giao thức và xử lý nội dung. Xử lý giao thức có nghĩa là quan tâm đến sự tương tác giữa máy khách và máy chủ: tạo yêu cầu ở định dạng chính xác, diễn giải các tiêu đề quay lại với dữ liệu, xác nhận rằng dữ liệu đã được nhận, v.v. Xử lý nội dung có nghĩa là chuyển đổi dữ liệu thô dữ liệu thành một định dạng mà Java có thể hiểu được, chẳng hạn như Dòng đầu vào hoặc Đoạn âm thanh. Hai vấn đề này, giao thức xử lý và nội dung xử lý, là khác biệt. Phần mềm hiển thị ảnh GIF không quan tâm liệu ảnh có được truy xuất qua FTP, HTTP, gopher hay một số giao thức mới hay không. Tương tự như vậy, trình xử lý giao thức, quản lý kết nối và tương tác với máy chủ, không quan tâm xem nó đang nhận tệp HTML hay tệp phim MPEG; tối đa, nó sẽ trích xuất một loại nội dung từ các tiêu đề để chuyển cho trình xử lý nội dung. Java chia nhiệm vụ xử lý các giao thức thành một số phần. Kết quả là không có một lớp nào được gọi là ProtocolHandler. Thay vào đó, các phần của cơ chế xử lý giao thức được thực hiện bởi bốn lớp khác nhau trong gói java.net: URL, URLStreamHandler, URLConnection và URLStreamHandlerFactory. URL là lớp cụ thể duy nhất trong nhóm này; URLStreamHandler và URLConnection đều là các lớp trừu tượng và URLStreamHandlerFactory là một giao diện. Do đó, nếu bạn định triển khai một trình xử lý giao thức mới, bạn phải viết các lớp con cụ thể cho URLStreamHandler và URLConnection. Để sử dụng các lớp này, bạn cũng có thể phải viết một lớp triển khai giao diện URLStreamHandlerFactory.

16.1 What Is a Protocol Handler? Xử lý giao thức là gì?

The way the URL, URLStreamHandler , URLConnection, and URLStreamHandlerFactory classes work together can be confusing. Everything starts with a URL, which represents a pointer to a particular Internet resource. Each URL specifies the protocol used to access the resource; typical values for the protocol include mailto, http, and ftp. When you construct a URL object from the URL's string representation, the constructor strips the protocol field and passes it to theURLStreamHandlerFactory. The factory's job is to take the protocol, locate the right subclass of URLStreamHandler for the protocol, and create a new instance of that stream handler, which is stored as a field within the URL object. Each application has at most one URLStreamHandlerFactory; once the factory has been installed, attempting to install another will throw an Error.

Now that the URL object has a stream handler, it asks the stream handler to finish parsing the URL string and create a subclass of URLConnection that knows how to talk to servers using this protocol. URLStreamHandler subclasses andURLConnection subclasses always come in pairs; the stream handler for a protocol always knows how to find an appropriate URLConnection for its protocol. It is worth noting that the stream handler does most of the work of parsing the URL. The format of the URL, although it is standard, depends on the protocol; therefore, it must be parsed by a URLStreamHandler, which knows about a particular protocol, and not by the URL object, which is generic and thus should have no knowledge of specific protocols. This also means that if you are writing a new stream handler, you can define a new URL format that's appropriate to your task.

The URLConnection class, which you learned about in the previous chapter, represents an active connection to the Internet resource. It is responsible for interacting with the server. A URLConnection knows how to generate requests and interpret the headers that the server returns. The output from a URLConnection is the raw data requested with all traces of the protocol (headers, etc.) stripped, ready for processing by a content handler.

In most applications, you don't need to worry about URLConnection objects and stream handlers; they are hidden by the URL class, which provides a simple interface to the methods you need. When you call the getInputStream( ) ,getOutputStream( ), and getContent( ) methods of the URL class, you are really calling similarly named methods in the URLConnection class. We have seen that interacting directly with a URLConnection can be convenient when you need a little

more control over communication with a server, most commonly when downloading binary files.

However, the URLConnection and URLStreamHandler classes are even more important when you need to add new protocols. By writing subclasses of these classes, you can add support for standard protocols such as finger, whois, or NTP that Java doesn't support out of the box. Furthermore, you're not limited to established protocols with well-known services. You can create new protocols that perform database queries, search across multiple Internet search engines, view pictures from binary newsgroups, and more. You can add new kinds of URLs as needed to represent the new types of resources. Furthermore, Java applications can be built so that they can load new protocol handlers at runtime. Unlike current browsers such as Mozilla and Internet Explorer, which contain explicit knowledge of all the protocols and content types they can handle, a Java browser can be a relatively lightweight skeleton that loads new handlers as needed. Supporting a new protocol just means adding some

new classes in predefined locations, not writing an entirely new release of the browser.

What's involved in adding support for a new protocol? As I said earlier, you need to write two new classes: a subclass of URLConnection and a subclass of URLStreamHandler. You may also need to write a class that implements theURLStreamHandlerFactory interface. Your URLConnection subclass handles the interaction with the server, converts anything the server sends into an InputStream, and converts anything the client sends into an OutputStream. This subclass must implement the abstract method connect( ); it may also override the concrete methods getInputStream( ), getOutputStream( ), and getContent Type( ).

The URLStreamHandler subclass parses the string representation of the URL into its separate parts and creates a new URLConnection object that understands that URL's protocol. This subclass must implement the abstract openConnection( ) method, which returns the new URLConnection to its caller. If the String representation of the URL doesn't look like a standard http URL, then you should also override the parseURL( ) and toExternalForm( ) methods.

Finally, you may need to create a class that implements the URLStreamHandlerFactory interface. The URLStreamHandlerFactory helps the application find the right protocol handler for each type of URL. TheURLStreamHandlerFactory interface has a single method, create URLStreamHandler( ), which returns a URLStreamHandler object. This method

must find the appropriate subclass of URLStreamHandler given only the protocol (e.g., ftp); that is, it must understand whatever package and class naming conventions you use for your stream handlers. Since URLStreamHandlerFactory is an interface, you can place your createURLStreamHandler( ) method in any convenient class, perhaps the main class of your application.

When it first encounters a protocol, Java looks for URLStreamHandler classes in this order:

1. First, Java checks to see whether a URLStreamHandlerFactory is installed. If it is, the factory is asked for a URLStreamHandler for the protocol.

1. If a URLStreamHandlerFactory isn't installed or if Java can't find a URLStreamHandler for the protocol, then Java looks in the packages named in the java.protocol.handler.pkgs system property for a sub-package that shares the protocol name and a class called Handler. The value of this property is a list of package names separated by a vertical bar (|). Thus, to indicate that Java should seek protocol handlers in the com.macfaq.net.www and org.cafeaulait.protocols packages, you would add this line to your properties file:  
   java.protocol.handler.pkgs=com.macfaq.net.www|org.cafeaulait.pr otocols  
   Then to find an FTP protocol handler (for example), Java would look first for the class com.macfaq.net.www.ftp.Handler. If that weren't found, Java would next try to instantiate org.cafeaulait.protocols.ftp.Handler.
2. Finally, if all else fails, Java looks for a URLStreamHandler named sun.net.www.protocol.name.Handler, where *name* is replaced by the name of the protocol; for example, sun.net.www.protocol.ftp.Handler.

In the early days of Java (circa 1995) Sun was promising that protocols could be installed at runtime from the server that used them. For instance, in 1996, James Gosling and Henry McGilton wrote: "The HotJava Browser is given a reference to an object (a URL). If the handler for that protocol is already loaded, it will be used. If not, the HotJava Browser will search first the local system and then the system that is the target of the URL." [1] However, the loading of protocol handlers from web sites was never implemented; and Sun doesn't much talk about it anymore.

[1] James Gosling and Henry McGilton, The Java Language Environment, A White Paper, May 1996, http://java.sun.com/docs/white/langenv/HotJava.doc1.html.

Most of the time, an end user who wants to permanently install an extra protocol handler in a program such as HotJava will place the necessary classes in the program's class path and add the package prefix to the java.protocol.handler.pkgs property. However, a programmer who just wants to add a custom proto col handler to her program at compile time will write and install a URLStreamHandlerFactory that knows how to find her custom protocol handlers. The factory can tell an application to look for URLStreamHandler classes in any place that's convenient: on a web site, in the same directory as the application, or somewhere in the user's class path.

When each of these classes has been written and compiled, you're ready to write an application that uses your new protocol handler. Assuming that you're using a URLStreamHandlerFactory, pass the factory object to the static URL.setURL StreamHandlerFactory( ) method like this:

URL.setURLStreamHandlerFactory(new MyURLStreamHandlerFactory( ));

This method can be called only once in the lifetime of an application. If it is called a second time, it will throw an Error. Untrusted applets will generally not be allowed to install factories or change the java.protocol.handler.pkgs property. Consequently, protocol handlers are primarily of use to standalone applications such as HotJava; Netscape and Internet Explorer use their own native C code instead of Java to handle protocols, so they're limited to a fixed set of protocols.

To summarize, here's the sequence of events:

1. The program constructs a URL object.
2. The constructor uses the arguments it's passed to determine the protocol part  
   of the URL, e.g., http.
3. The URL( ) constructor tries to find a URLStreamHandler for the given  
   protocol like this:
   * If the protocol has been used before, then the URLStreamHandler  
     object is retrieved from a cache.
   * Otherwise, if a URLStreamHandlerFactory has been set, then the  
     protocol string is passed to the factory's createURLStreamHandler( )  
     method.
   * If the protocol hasn't been seen before and there's no URlStream  
     HandlerFactory, then the constructor attempts to instantiate a URLStreamHandler object named *protocol*.Handler in one of the packages listed in the java.protocol.handler.pkgs property.
   * Failing that, the constructor attempts to instantiate a URLStreamHandler object named *protocol*.Handler in the sun.net.www.protocol package.
   * If any of these attempts succeed in retrieving a URLStreamHandler object, the URL constructor sets the URL object's handler field. If none of the attempts succeed, the constructor throws a MalformedURLException.
4. The program calls the URL object's openConnection( ) method.
5. The URL object asks the URLStreamHandler to return a URLConnection object  
   appropriate for this URL. If there's any problem, an IOException is thrown.  
   Otherwise, a URLConnection object is returned.
6. The program uses the methods of the URLConnection class to interact with the  
   remote resource.

Instead of calling openConnection( ) in step 4, the program can call getContent( ) or getInputStream( ). In this case, the URLStreamHandler still instantiates a URLConnection object of the appropriate class. However, instead of returning the URLConnection object itself, the URLStreamHandler returns the result of URLConnection's getContent( ) or getInputStream( ) method.

Cách các lớp URL, URLStreamHandler, URLConnection và URLStreamHandlerFactory hoạt động cùng nhau có thể gây nhầm lẫn. Mọi thứ bắt đầu với một URL, đại diện cho một con trỏ đến một tài nguyên Internet cụ thể. Mỗi URL chỉ định giao thức được sử dụng để truy cập tài nguyên; các giá trị điển hình cho giao thức bao gồm mailto, http và ftp. Khi bạn tạo một đối tượng URL từ biểu diễn chuỗi của URL, hàm tạo sẽ tách trường giao thức và chuyển nó đến theURLStreamHandlerFactory. Công việc của nhà máy là lấy giao thức, định vị lớp con phù hợp của URLStreamHandler cho giao thức và tạo một phiên bản mới của trình xử lý luồng đó, được lưu trữ dưới dạng một trường trong đối tượng URL. Mỗi ứng dụng có nhiều nhất một URLStreamHandlerFactory; khi nhà máy đã được cài đặt, việc cố gắng cài đặt nhà máy khác sẽ xuất hiện một Lỗi. Bây giờ đối tượng URL có một trình xử lý luồng, nó yêu cầu trình xử lý luồng hoàn tất việc phân tích chuỗi URL và tạo một lớp con của URLConnection biết cách nói chuyện với các máy chủ sử dụng giao thức này. Các lớp con URLStreamHandler và các lớp conURLConnection luôn đi theo cặp; trình xử lý luồng cho một giao thức luôn biết cách tìm một URLConnection thích hợp cho giao thức của nó. Cần lưu ý rằng trình xử lý luồng thực hiện hầu hết công việc phân tích cú pháp URL. Định dạng của URL, mặc dù nó là tiêu chuẩn, phụ thuộc vào giao thức; do đó, nó phải được phân tích cú pháp bởi một URLStreamHandler, biết về một giao thức cụ thể, chứ không phải bởi đối tượng URL, là đối tượng chung chung và do đó không được biết về các giao thức cụ thể. Điều này cũng có nghĩa là nếu bạn đang viết một trình xử lý luồng mới, bạn có thể xác định một định dạng URL mới phù hợp với nhiệm vụ của mình. Lớp URLConnection, mà bạn đã học trong chương trước, đại diện cho một kết nối đang hoạt động với tài nguyên Internet. Nó có nhiệm vụ tương tác với máy chủ. Một URLConnection biết cách tạo các yêu cầu và diễn giải các tiêu đề mà máy chủ trả về. Kết quả đầu ra từ URLConnection là dữ liệu thô được yêu cầu với tất cả các dấu vết của giao thức (tiêu đề, v.v.) bị tước bỏ, sẵn sàng để xử lý bởi trình xử lý nội dung. Trong hầu hết các ứng dụng, bạn không cần phải lo lắng về các đối tượng URLConnection và trình xử lý luồng; chúng bị ẩn bởi lớp URL, lớp này cung cấp một giao diện đơn giản cho các phương thức bạn cần. Khi bạn gọi các phương thức getInputStream (), getOutputStream () và getContent () của lớp URL, bạn thực sự đang gọi các phương thức có tên tương tự trong lớp URLConnection. Chúng tôi nhận thấy rằng việc tương tác trực tiếp với URLConnection có thể thuận tiện khi bạn cần một chút kiểm soát nhiều hơn giao tiếp với máy chủ, phổ biến nhất là khi tải xuống các tệp nhị phân. Tuy nhiên, các lớp URLConnection và URLStreamHandler thậm chí còn quan trọng hơn khi bạn cần thêm các giao thức mới. Bằng cách viết các lớp con của các lớp này, bạn có thể thêm hỗ trợ cho các giao thức chuẩn như ngón tay, whois hoặc NTP mà Java không hỗ trợ. Hơn nữa, bạn không bị giới hạn ở các giao thức đã thiết lập với các dịch vụ nổi tiếng. Bạn có thể tạo các giao thức mới để thực hiện các truy vấn cơ sở dữ liệu, tìm kiếm trên nhiều công cụ tìm kiếm trên Internet, xem ảnh từ các nhóm tin nhị phân, v.v. Bạn có thể thêm các loại URL mới nếu cần để đại diện cho các loại tài nguyên mới. Hơn nữa, các ứng dụng Java có thể được xây dựng để chúng có thể tải các trình xử lý giao thức mới trong thời gian chạy. Không giống như các trình duyệt hiện tại như Mozilla và Internet Explorer, chứa kiến ​​thức rõ ràng về tất cả các giao thức và kiểu nội dung mà chúng có thể xử lý, trình duyệt Java có thể là một bộ khung tương đối nhẹ có thể tải các trình xử lý mới khi cần thiết. Hỗ trợ một giao thức mới chỉ có nghĩa là thêm một số các lớp mới ở các vị trí được xác định trước, không phải viết một bản phát hành hoàn toàn mới của trình duyệt. Điều gì liên quan đến việc thêm hỗ trợ cho một giao thức mới? Như tôi đã nói trước đó, bạn cần viết hai lớp mới: một lớp con của URLConnection và một lớp con của URLStreamHandler. Bạn cũng có thể cần viết một lớp triển khai giao diệnURLStreamHandlerFactory. Lớp con URLConnection của bạn xử lý tương tác với máy chủ, chuyển đổi bất cứ thứ gì máy chủ gửi thành InputStream và chuyển đổi bất cứ thứ gì mà máy khách gửi thành OutputStream. Lớp con này phải triển khai phương thức trừu tượng connect (); nó cũng có thể ghi đè các phương thức cụ thể getInputStream (), getOutputStream () và getContent Type (). Lớp con URLStreamHandler phân tích trình bày chuỗi của URL thành các phần riêng biệt của nó và tạo một đối tượng URLConnection mới hiểu được giao thức của URL đó. Lớp con này phải triển khai phương thức trừu tượng openConnection (), phương thức này trả về URLConnection mới cho trình gọi của nó. Nếu biểu diễn chuỗi của URL không giống như URL http chuẩn, thì bạn cũng nên ghi đè các phương thức parseURL () và toExternalForm (). Cuối cùng, bạn có thể cần tạo một lớp triển khai giao diện URLStreamHandlerFactory. URLStreamHandlerFactory giúp ứng dụng tìm đúng

16.2 The URLStreamHandler Class

The abstract URLStreamHandler class is a superclass for classes that handle specific protocols—for example, HTTP. You rarely call the methods of the URLStreamHandler class directly; they are called by other methods in the URL andURLConnection classes. By overriding the URLStreamHandler methods in your own

subclass, you teach the URL class how to handle new protocols. Therefore, we'll focus on overriding the methods of URLStreamHandler rather than on calling the methods.

Lớp URLStreamHandler trừu tượng là lớp cha dành cho các lớp xử lý các giao thức cụ thể — ví dụ: HTTP. Bạn hiếm khi gọi trực tiếp các phương thức của lớp URLStreamHandler; chúng được gọi bằng các phương thức khác trong các lớp URL vàURLConnection. Bằng cách ghi đè các phương thức URLStreamHandler của riêng bạn lớp con, bạn dạy lớp URL cách xử lý các giao thức mới. Do đó, chúng tôi sẽ tập trung vào việc ghi đè các phương thức của URLStreamHandler hơn là gọi các phương thức.

16.2.1 The Constructor

You do not create URLStreamHandler objects directly. Instead, when a URL is constructed with a protocol that hasn't been seen before, Java asks the application's URLStreamHandlerFactory to create the appropriate URLStreamHandler subclass for the protocol. If that fails, Java guesses at the fully package-qualified name of the URLStreamHandler class and uses Class.forName( ) to attempt to construct such an object. This means concrete subclasses should have a noargs constructor. The single constructor for URLStreamHandler doesn't take any arguments:

public URLStreamHandler( )

Because URLStreamHandler is an abstract class, this constructor is never called

directly; it is only called from the constructors of subclasses.

Bạn không tạo trực tiếp các đối tượng URLStreamHandler. Thay vào đó, khi một URL được tạo bằng một giao thức chưa từng thấy trước đây, Java sẽ yêu cầu URLStreamHandlerFactory của ứng dụng tạo lớp con URLStreamHandler thích hợp cho giao thức. Nếu không thành công, Java sẽ đoán ở tên đầy đủ đủ điều kiện gói của lớp URLStreamHandler và sử dụng Class.forName () để cố gắng xây dựng một đối tượng như vậy. Điều này có nghĩa là các lớp con bê tông phải có một hàm tạo noargs. Hàm tạo duy nhất cho URLStreamHandler không nhận bất kỳ đối số nào: URL công khaiStreamHandler () Vì URLStreamHandler là một lớp trừu tượng, hàm tạo này không bao giờ được gọi trực tiếp; nó chỉ được gọi từ các hàm tạo của các lớp con.

16.2.2 Methods for Parsing URLs - Phương thức chuyển URL

The first responsibility of a URLStreamHandler is to split a string representation of a URL into its component parts and use those parts to set the various fields of the URL object. The parseURL( ) method splits the URL into parts, possibly using setURL( ) to assign values to the URL's fields. It is very difficult to imagine a situation in which you would call parseURL( ) directly; instead, you override it to change the behavior of the URL class.

Trách nhiệm đầu tiên của URLStreamHandler là chia biểu diễn chuỗi của một URL thành các phần thành phần của nó và sử dụng các phần đó để đặt các trường khác nhau của đối tượng URL. Phương thức parseURL () chia URL thành các phần, có thể sử dụng setURL () để gán giá trị cho các trường của URL. Rất khó để hình dung một tình huống mà bạn sẽ gọi trực tiếp parseURL (); thay vào đó, bạn ghi đè nó để thay đổi hành vi của lớp URL.

16.2.2.1 Protected void parseURL(URL u, String spec, int start, int limit)

This method parses a string spec into a URL u. All characters in the spec string before start should already have been parsed into the URL u. Characters after limit are ignored. Generally, the protocol will have already been parsed and stored in u before this method is invoked, and start will be adjusted so that it starts with the character after the colon that delimits the protocol.

The task of parseURL( ) is to set the URL u's protocol, host, port, file, and ref fields. It can assume that any parts of the String that are before start and after limit have already been parsed or can be ignored.

The parseURL( ) method that Java supplies assumes that the URL looks more or less like an http URL:

protocol://www.host.com:port/directory/another\_directory/file#ref

This works for ftp and gopher URLs. It does not work for mailto or news URLs and may not be appropriate for any new URL types you define. If your protocol handler uses URLs that fit this form, you don't have to override parseURL( ) at all; the method inherited from URLStreamHandler will work just fine. If your URLs are completely different, you must supply a parseURL( ) method that parses the URL completely. However, there's often a middle ground that can make your task easier. If

your URL looks somewhat like a standard URL, you can implement a parseURL( ) method that handles the nonstandard portion of your URL and then calls super.parseURL( ) to do the rest of the work, setting the offset and limitarguments to indicate the portion of the URL that you didn't parse.

For example, a mailto URL looks like mailto:elharo@metalab.unc.edu. First, you need to figure out how to map this into the URL class's protocol, host, port, file, and ref fields. The protocol is clearly mailto. Everything after the @ can be the host. The hard question is what to do with the username. Since a mailto URL really doesn't have a file portion, we will use the URL class's file field to hold the username. The ref can be set to the empty string or null. The parseURL( ) method that follows implements this scheme:

Phương thức này phân tích cú pháp một chuỗi đặc tả thành một URL u. Tất cả các ký tự trong chuỗi đặc tả trước khi bắt đầu đã được phân tích cú pháp thành URL u. Các ký tự sau khi giới hạn bị bỏ qua. Nói chung, giao thức sẽ đã được phân tích cú pháp và lưu trữ trong u trước khi phương thức này được gọi và khởi động sẽ được điều chỉnh để nó bắt đầu bằng ký tự sau dấu hai chấm phân định giao thức. Nhiệm vụ của parseURL () là đặt các trường giao thức, máy chủ, cổng, tệp và tham chiếu của URL u. Nó có thể giả định rằng bất kỳ phần nào của Chuỗi trước khi bắt đầu và sau giới hạn đã được phân tích cú pháp hoặc có thể bị bỏ qua. Phương thức parseURL () mà Java cung cấp giả định rằng URL trông giống hoặc ít hơn một URL http: giao thức: //www.host.com: port / directory / another\_directory / file # ref Điều này hoạt động cho các URL ftp và gopher. Nó không hoạt động cho các URL mailto hoặc tin tức và có thể không thích hợp cho bất kỳ loại URL mới nào mà bạn xác định. Nếu trình xử lý giao thức của bạn sử dụng URL phù hợp với biểu mẫu này, bạn không phải ghi đè parseURL (); phương thức được kế thừa từ URLStreamHandler sẽ hoạt động tốt. Nếu các URL của bạn hoàn toàn khác nhau, bạn phải cung cấp phương thức parseURL () để phân tích cú pháp URL hoàn toàn. Tuy nhiên, thường có một nền tảng trung gian có thể làm cho nhiệm vụ của bạn dễ dàng hơn. Nếu URL của bạn trông giống như một URL chuẩn, bạn có thể triển khai phương thức parseURL () để xử lý phần không chuẩn của URL và sau đó gọi super.parseURL () để thực hiện phần còn lại của công việc, đặt đối số bù và giới hạn để chỉ ra phần của URL mà bạn không phân tích cú pháp. Ví dụ: một URL mailto trông giống như mailto: elharo@metalab.unc.edu. Trước tiên, bạn cần tìm cách ánh xạ điều này vào các trường giao thức, máy chủ, cổng, tệp và tham chiếu của lớp URL. Giao thức rõ ràng là mailto. Mọi thứ sau ký tự @ đều có thể là máy chủ lưu trữ. Câu hỏi khó là phải làm gì với tên người dùng. Vì URL mailto thực sự không có phần tệp, chúng tôi sẽ sử dụng trường tệp của lớp URL để giữ tên người dùng. Tham chiếu có thể được đặt thành chuỗi rỗng hoặc null. Phương thức parseURL () theo sau sẽ triển khai lược đồ này:

public void parseURL(URL u, String spec, int start, int limit) {

String protocol = u.getProtocol( );

String host = "";

int port = u.getPort( );

String file = ""; // really username

String ref = null;

if( start < limit) {

String address = spec.substring(start, limit);

int atSign = address.indexOf('@');

if (atSign >= 0) {

host = address.substring(atSign+1);

file = address.substring(0, atSign);

}

}

this.setURL(u, protocol, host, port, file, ref);

}

Rather than borrowing an unused field from the URL object, it's possibly a better idea to store protocol-specific parts of the URL, such as the username, in fields of the URLStreamHandler subclass. The disadvantage of this approach is that such fields can be seen only by your own code; in this example, you couldn't use the getFile( ) method in the URL class to retrieve the username. Here's a version of parseURL( ) that stores the username in a field of the Handler subclass. When the connection is opened, the username can be copied into the Mailto URLConnection object that results. That class would provide some sort of getUserName( ) method:

Thay vì mượn một trường không sử dụng từ đối tượng URL, có thể là một ý tưởng tốt hơn để lưu trữ các phần cụ thể của giao thức của URL, chẳng hạn như tên người dùng, trong các trường của lớp con URLStreamHandler. Nhược điểm của cách tiếp cận này là các trường như vậy chỉ có thể được nhìn thấy bằng mã của riêng bạn; trong ví dụ này, bạn không thể sử dụng phương thức getFile () trong lớp URL để truy xuất tên người dùng. Đây là phiên bản của parseURL () lưu trữ tên người dùng trong một trường của lớp con Trình xử lý. Khi kết nối được mở, tên người dùng có thể được sao chép vào đối tượng Mailto URLConnection. Lớp đó sẽ cung cấp một số loại phương thức getUserName ():

**String username = "";**

public void parseURL(URL u, String spec, int start, int limit) {

String protocol = u.getProtocol( );

String host = "";

int port = u.getPort( );

String file = "";

String ref = null;

if( start < limit) {

String address = spec.substring(start, limit);

int atSign = address.indexOf('@');

if (atSign >= 0) {

host = address.substring(atSign+1);

**this.username = address.substring(0, atSign);**

} }

this.setURL(u, protocol, host, port, file, ref);

}

16.2.2.2 Protected String toExternalForm(URL u)

This method puts the pieces of the URL u—that is, its protocol, host, port, file, and ref fields—back together in a String. If you override parseURL( ), you should also override toExternalForm( ) . Here's a toExternalForm( ) method for a mailto URL; it assumes that the username has been stored in the URL's host field:

Phương thức này đặt các phần của URL u — nghĩa là, các trường giao thức, máy chủ, cổng, tệp và tham chiếu của nó — trở lại với nhau trong một Chuỗi. Nếu bạn ghi đè parseURL (), bạn cũng nên ghi đè thànhExternalForm (). Đây là một phương thức toExternalForm () cho một URL mailto; nó giả định rằng tên người dùng đã được lưu trữ trong trường máy chủ của URL:

protected String toExternalForm(URL u) {

return "mailto:" + u.getFile() + "@" + u.getHost( );

}

Since toExternalForm( ) is protected, you probably won't call this method directly. However, it is called by the public toExternalForm( ) and toString( ) methods of the URL class, so any change you make here is reflected when you convert URL objects to strings.

Vì toExternalForm () được bảo vệ, có thể bạn sẽ không gọi phương thức này trực tiếp. Tuy nhiên, nó được gọi bởi các phương thức public toExternalForm () và toString () của lớp URL, vì vậy bất kỳ thay đổi nào bạn thực hiện ở đây đều được phản ánh khi bạn chuyển đổi các đối tượng URL thành chuỗi.

16.2.2.3 Protected void setURL(URL u, String protocol, String host, int port, String file, String ref )

This method sets the host, port, file, and ref fields of the URL u to the given values. It sets the protocol field to the protocol argument in Java 1.1.x and earlier only. (In Java 1.2 and later, this field is set before this method is called and cannot be changed. The protocol argument is ignored.) This method is used by parseURL( ) to set these fields to the values it has found by parsing the URL. You need to call this method at the end of the parseURL( ) method when you subclass URLStreamHandler.

This method is deprecated in Java 1.3 (though not in Java 1.2). The reason is that Java 1.3 also allows you to add a query string, user info, and an authority to the URL. Thus, Java 1.3 prefers this alternative setURL( ) method, which supports those features:

Phương thức này đặt các trường máy chủ, cổng, tệp và tham chiếu của URL u thành các giá trị đã cho. Nó đặt trường giao thức thành đối số giao thức chỉ trong Java 1.1.x trở về trước. (Trong Java 1.2 trở lên, trường này được đặt trước khi phương thức này được gọi và không thể thay đổi được. Đối số giao thức bị bỏ qua.) Phương thức này được parseURL () sử dụng để đặt các trường này thành các giá trị mà nó đã tìm thấy bằng cách phân tích cú pháp URL . Bạn cần gọi phương thức này ở cuối phương thức parseURL () khi bạn phân lớp URLStreamHandler. Phương thức này không được chấp nhận trong Java 1.3 (mặc dù không có trong Java 1.2). Lý do là Java 1.3 cũng cho phép bạn thêm chuỗi truy vấn, thông tin người dùng và quyền vào URL. Do đó, Java 1.3 thích phương thức setURL () thay thế này, hỗ trợ các tính năng đó:

protected void setURL(URL u, String protocol, String host, int port,

// Java 1.3

String authority, String userInfo, String path, String query,

String ref)

In Java 1.2 and earlier, user info and authority can be included as part of the host argument. The query string can be included at the end of the file. Even in Java 1.3, this method is a little flaky, since the host, port, and user info together make up the authority. In the event of a conflict between them, they're all stored separately, but the host, port, and user info are used in preference to the authority when deciding which site to connect to.

This is actually quite relevant to the mailto example, since mailto URLs often have query strings that indicate the subject or other header; for example, mailto:elharo@metalab.unc.edu?subject=JavaReading. Here the query string issubject=JavaReading. If we were to rewrite the above parseURL( ) method to support mailto URLs in this format, the result would look like this:

Trong Java 1.2 trở về trước, thông tin người dùng và quyền hạn có thể được đưa vào như một phần của đối số máy chủ. Chuỗi truy vấn có thể được đưa vào cuối tệp. Ngay cả trong Java 1.3, phương pháp này cũng hơi phức tạp, vì máy chủ, cổng và thông tin người dùng cùng tạo nên quyền hạn. Trong trường hợp có xung đột giữa chúng, tất cả chúng đều được lưu trữ riêng biệt, nhưng máy chủ lưu trữ, cổng và thông tin người dùng được sử dụng tùy theo cơ quan có thẩm quyền khi quyết định kết nối với trang web nào. Điều này thực sự khá liên quan đến ví dụ mailto, vì URL mailto thường có các chuỗi truy vấn cho biết chủ đề hoặc tiêu đề khác; ví dụ: mailto: elharo@metalab.unc.edu? subject = JavaReading. Đây là chuỗi truy vấn Issubject = JavaReading. Nếu chúng tôi viết lại phương thức parseURL () ở trên để hỗ trợ các URL mailto ở định dạng này, kết quả sẽ như sau:

public void parseURL(URL u, String spec, int start, int limit) {

String protocol = u.getProtocol( );

String host

int port

String file

String userInfo = null;

String query = null;

String ref = null;

= "";

= u.getPort( );

= "";

if (start < limit) {

String address = spec.substring(start, limit);

int atSign = address.indexOf('@');

int questionMark = address.indexOf('?');

int hostEnd = questionMark >= 0 ? questionMark :

address.length( );

if (atSign >= 0) {

host = address.substring(atSign+1, hostEnd);

userInfo = address.substring(0, atSign);

}

if (questionMark >= 0 && questionMark > atSign) {

query = address.substring(questionMark + 1);

}

}

String authority = "";

if (userInfo != null) authority += userInfo + '@';

authority += host;

if (port >= 0) authority += ":" + port;

this.setURL(u, protocol, host, port, authority, userInfo, file,

query, ref);

}

Note that this works only in Java 1.3 and later. It will not work in Java 1.1 or 1.2.

16.2.2.4 Protected int getDefaultPort( ) // Java 1.3

Java 1.3 adds a getDefaultPort( ) method to the URLStreamHandler class whose responsibility is to return the default port for the protocol; e.g., 80 for HTTP. The default implementation of this method simply returns -1, but each subclass should override that with the appropriate default port for the protocol it handles. For example, here's a getDefaultPort( ) method for the finger protocol that normally operates on port 79:

Java 1.3 thêm một phương thức getDefaultPort () vào lớp URLStreamHandler có trách nhiệm trả lại cổng mặc định cho giao thức; ví dụ: 80 cho HTTP. Việc triển khai mặc định của phương thức này chỉ trả về -1, nhưng mỗi lớp con sẽ ghi đè lên đó bằng cổng mặc định thích hợp cho giao thức mà nó xử lý. Ví dụ: đây là phương thức getDefaultPort () cho giao thức ngón tay thường hoạt động trên cổng 79:

public int getDefaultPort( ) {

return 79; }

As well as providing the right port for finger, this overriding method also makes getDefaultPort( ) public. Although there's only a default implementation of this method in Java 1.3, there's no reason you can't provide it in your own subclasses in any version of Java. You simply won't be able to invoke it polymorphically from a reference typed as the superclass.

Ngoài việc cung cấp cổng phù hợp cho ngón tay, phương pháp ghi đè này cũng đặt getDefaultPort () ở chế độ công khai. Mặc dù chỉ có một triển khai mặc định của phương thức này trong Java 1.3, không có lý do gì bạn không thể cung cấp nó trong các lớp con của riêng mình trong bất kỳ phiên bản Java nào. Đơn giản là bạn sẽ không thể gọi nó một cách đa hình từ một tham chiếu được đánh là lớp cha.

16.2.2.5 Protected InetAddress getHostAddress(URL u) // Java 1.3

Java 1.3 also adds a getHostAddress( ) method to the URLStreamHandler class whose responsibility is to return an InetAddress object pointing to the server in the URL. This requires a DNS lookup, and the method does block while the lookup is made. However, it does not throw any exceptions. If the host can't be located, whether because the URL does not contain a host part or there is a DNS failure or a SecurityException, then it simply returns null. The default implementation of this method is sufficient for any reasonable case. It shouldn't be necessary to override it.

Java 1.3 cũng thêm một phương thức getHostAddress () vào lớp URLStreamHandler có trách nhiệm trả về một đối tượng InetAddress trỏ đến máy chủ trong URL. Điều này yêu cầu tra cứu DNS và phương pháp này sẽ chặn trong khi thực hiện tra cứu. Tuy nhiên, nó không ném bất kỳ ngoại lệ nào. Nếu không thể định vị máy chủ lưu trữ, cho dù do URL không chứa phần máy chủ lưu trữ hoặc có lỗi DNS hoặc SecurityException, thì nó chỉ trả về null. Việc triển khai mặc định của phương pháp này là đủ cho mọi trường hợp hợp lý. Không cần thiết phải ghi đè nó.

16.2.2.6 Protected boolean hostsEqual(URL u1, URL u2) // Java 1.3

Java 1.3's hostsEqual( ) determines whether the two URLs refer to the same server. This method attempts to use DNS to actually look up the hosts. If that succeeds for both hosts, then it can tell that, for example, http://metalab.unc.edu/Dave/Dr- Fun/latest.jpg and ftp://sunsite.unc.edu/pub/linux/distributions/redhat/current/ are the same host. However, if the DNS lookup fails for any reason, then hostsEqual( ) falls back to a simple case-insensitive string comparison, in which case it would think those were two different hosts.

The default implementation of this method is sufficient for most cases. You probably won't need to override it. The only case I can imagine where you might want to is if you were trying to make mirror sites on different servers appear equal.

Java 1.3's hostsEqual () xác định xem hai URL có tham chiếu đến cùng một máy chủ hay không. Phương pháp này cố gắng sử dụng DNS để thực sự tra cứu các máy chủ. Nếu điều đó thành công cho cả hai máy chủ, thì nó có thể nói rằng, ví dụ: http://metalab.unc.edu/Dave/Dr- Fun / new.jpg và ftp://sunsite.unc.edu/pub/linux/ các bản phân phối / redhat / current / là cùng một máy chủ. Tuy nhiên, nếu tra cứu DNS không thành công vì bất kỳ lý do gì, thì hostsEqual () sẽ trở lại so sánh chuỗi đơn giản không phân biệt chữ hoa chữ thường, trong trường hợp đó nó sẽ nghĩ rằng đó là hai máy chủ khác nhau. Việc triển khai mặc định của phương pháp này là đủ cho hầu hết các trường hợp. Bạn có thể sẽ không cần ghi đè nó. Trường hợp duy nhất tôi có thể tưởng tượng nơi bạn có thể muốn là nếu bạn đang cố gắng làm cho các trang nhân bản trên các máy chủ khác nhau trông giống nhau.

16.2.2.7 Protected boolean sameFile(URL u1, URL u2) // Java 1.3

Java 1.3's sameFile( ) determines whether two URLs point to the same file. It does this by comparing the protocol, host, port, and path. The files are considered to be the same only if each of those four pieces is the same. However, it does not consider the query string or the ref. Furthermore, the hosts are compared by the hostsEqual( ) method so that metalab.unc.edu and sunsite.unc.edu can be recognized as the same if DNS can resolve them. This is similar to the sameFile( ) method of the URL class. Indeed, in Java 1.3, that sameFile( ) method just calls this sameFile( ) method.

The default implementation of this method is sufficient for most cases. You probably won't need to override it. You might perhaps want to do so though if you needed a more sophisticated test that converted paths to canonical paths or followed redirects before determining whether two URLs had the same file part.

Java 1.3's sameFile () xác định xem hai URL có trỏ đến cùng một tệp hay không. Nó thực hiện điều này bằng cách so sánh giao thức, máy chủ, cổng và đường dẫn. Các tệp chỉ được coi là giống nhau nếu mỗi phần trong số bốn phần đó giống nhau. Tuy nhiên, nó không xem xét chuỗi truy vấn hoặc tham chiếu. Hơn nữa, các máy chủ được so sánh bằng phương thức hostsEqual () để metalab.unc.edu và Sunite.unc.edu có thể được nhận dạng là giống nhau nếu DNS có thể phân giải chúng. Điều này tương tự với phương thức sameFile () của lớp URL. Thật vậy, trong Java 1.3, phương thức sameFile () đó chỉ gọi phương thức sameFile () này. Việc triển khai mặc định của phương pháp này là đủ cho hầu hết các trường hợp. Bạn có thể sẽ không cần ghi đè nó. Tuy nhiên, có lẽ bạn có thể muốn làm như vậy nếu bạn cần một thử nghiệm phức tạp hơn để chuyển đổi đường dẫn thành đường dẫn chuẩn hoặc chuyển hướng theo sau trước khi xác định xem hai URL có cùng một phần tệp hay không.

16.2.2.8 Protected boolean equals(URL u1, URL u2) // Java 1.3

The final equality method added by Java 1.3 tests almost the entire URL, including protocol, host, file, path, and ref. Only the query string is ignored. All five of these must be equal for the two URLs to be considered equal. Everything except the ref is

compared by the sameFile( ) method, so overriding that method changes the behavior of this one. The refs are compared by simple string equality. Since the sameFile( ) method uses hostsEqual( ) to compare hosts, this method does too. Thus it performs a DNS lookup if possible and may block. In Java 1.3, the equals( ) method of the URL class calls this method to compare two URL objects for equality. Again, you probably won't need to override this method. The default implementation should suffice for most purposes.

Phương thức bình đẳng cuối cùng được Java 1.3 thêm vào kiểm tra gần như toàn bộ URL, bao gồm giao thức, máy chủ, tệp, đường dẫn và ref. Chỉ chuỗi truy vấn bị bỏ qua. Tất cả năm điều này phải bằng nhau để hai URL được coi là bằng nhau. Tất cả mọi thứ ngoại trừ giới thiệu là được so sánh bởi phương thức sameFile (), vì vậy việc ghi đè phương thức đó sẽ thay đổi hành vi của phương thức này. Các tham chiếu được so sánh bằng đẳng thức chuỗi đơn giản. Vì phương thức sameFile () sử dụng hostsEqual () để so sánh các máy chủ, phương thức này cũng vậy. Do đó, nó thực hiện tra cứu DNS nếu có thể và có thể chặn. Trong Java 1.3, phương thức equals () của lớp URL gọi phương thức này để so sánh hai đối tượng URL cho bằng nhau. Một lần nữa, bạn có thể sẽ không cần ghi đè phương thức này. Việc triển khai mặc định sẽ đủ cho hầu hết các mục đích.

16.2.2.9 Protected int hashCode(URL u) // Java 1.3

Java 1.3 also gives URLStreamHandlers the opportunity to change the default hash code calculation by overriding this method. You should do this if you override equals( ), sameFile( ), or hostsEqual( ) to make sure that two equal URLobjects will have the same hash code, and two unequal URL objects will not have the same hash code, at least to a very high degree of probability.

Java 1.3 cũng cho URLStreamHandlers cơ hội thay đổi cách tính mã băm mặc định bằng cách ghi đè phương thức này. Bạn nên làm điều này nếu bạn ghi đè bằng equals (), sameFile () hoặc hostsEqual () để đảm bảo rằng hai đối tượng URL bằng nhau sẽ có cùng mã băm và hai đối tượng URL không bằng nhau sẽ không có cùng mã băm, ít nhất là đối với một mức độ xác suất rất cao.

16.2.3 A Method for Connecting

The second responsibility of a URLStreamHandler is to create a URLConnection object appropriate to the URL. This is done by the abstract openConnection( ) method.

Trách nhiệm thứ hai của URLStreamHandler là tạo một đối tượng URLConnection phù hợp với URL. Điều này được thực hiện bởi phương thức trừu tượng openConnection ().

16.2.3.1 Protected abstract URLConnection openConnection(URL u) throws IOException

This method must be overridden in each subclass of URLConnection. It takes a single argument u, which is the URL to connect to. It returns an unopened URLConnection, directed at the resource u points to. Each subclass of URLStreamHandler should know how to find the right subclass of URLConnection for the protocol it handles.

The openConnection( ) method is protected, so you usually do not call it directly; it is called by the openConnection( ) method of a URL class. The URL u that is passed as an argument is the URL that needs a connection. You override this method in your subclasses to handle a specific protocol. Your subclass's openConnection( ) method is usually extremely simple; in most cases, it just calls the constructor for the appropriate subclass of URLConnection. For example, URLStreamHandler for the mailto protocol might have an openConnection( ) method that looks like this:

Phương thức này phải được ghi đè trong mỗi lớp con của URLConnection. Nó cần một đối số u, là URL để kết nối. Nó trả về một URLConnection chưa mở, hướng đến tài nguyên mà bạn trỏ đến. Mỗi lớp con của URLStreamHandler phải biết cách tìm đúng lớp con của URLConnection cho giao thức mà nó xử lý. Phương thức openConnection () được bảo vệ, vì vậy bạn thường không gọi nó trực tiếp; nó được gọi bằng phương thức openConnection () của một lớp URL. URL u được chuyển làm đối số là URL cần kết nối. Bạn ghi đè phương thức này trong các lớp con của mình để xử lý một giao thức cụ thể. Phương thức openConnection () của lớp con của bạn thường cực kỳ đơn giản; trong hầu hết các trường hợp, nó chỉ gọi hàm tạo cho lớp con thích hợp của URLConnection. Ví dụ: URLStreamHandler cho giao thức mailto có thể có phương thức openConnection () giống như sau:

protected URLConnection openConnection(URL u) throws IOException {

return new com.macfaq.net.www.protocol.MailtoURLConnection(u);

}

Example 16.1. A mailto URLStreamHandler

package com.macfaq.net.www.protocol.mailto;

import java.net.\*;

import java.io.\*;

import java.util.\*;

public class Handler extends URLStreamHandler {

protected URLConnection openConnection(URL u) throws IOException {

return new MailtoURLConnection(u);

}

public void parseURL(URL u, String spec, int start, int limit) {

String protocol = u.getProtocol( );

String host

int port

String file

String userInfo = null;

String authority = null;

String query = null;

String ref = null;

= "";

= u.getPort( );

= ""; // really username

if( start < limit) {

String address = spec.substring(start, limit);

int atSign = address.indexOf('@');

if (atSign >= 0) {

host = address.substring(atSign+1);

file = address.substring(0, atSign);

}

}

// For Java 1.2 comment out this next line

this.setURL(u, protocol, host, port, authority,

userInfo, file, query, ref);

// In Java 1.2 and earlier uncomment the following line:

// this.setURL(u, protocol, host, port, file, ref);

}

protected String toExternalForm(URL u) {

return "mailto:" + u.getFile() + "@" + u.getHost( );;

} }

16.3 Writing a Protocol Handler

To demonstrate a complete protocol handler, let's write one for the finger protocol, which is defined in RFC 1288 and was introduced in Chapter 10. Finger is a relatively simple protocol compared to JDK-supported protocols such as HTTP and FTP. The client connects to port 79 on the server and sends a list of usernames followed by a carriage return/linefeed pair. The server responds with ASCII text containing information about each of the named users or, if no names were listed, a list of the currently logged in users. For example:

Phương thức này phải được ghi đè trong mỗi lớp con của URLConnection. Nó cần một đối số u, là URL để kết nối. Nó trả về một URLConnection chưa mở, hướng đến tài nguyên mà bạn trỏ đến. Mỗi lớp con của URLStreamHandler phải biết cách tìm đúng lớp con của URLConnection cho giao thức mà nó xử lý. Phương thức openConnection () được bảo vệ, vì vậy bạn thường không gọi nó trực tiếp; nó được gọi bằng phương thức openConnection () của một lớp URL. URL u được chuyển làm đối số là URL cần kết nối. Bạn ghi đè phương thức này trong các lớp con của mình để xử lý một giao thức cụ thể. Phương thức openConnection () của lớp con của bạn thường cực kỳ đơn giản; trong hầu hết các trường hợp, nó chỉ gọi hàm tạo cho lớp con thích hợp của URLConnection. Ví dụ: URLStreamHandler cho giao thức mailto có thể có phương thức openConnection () giống như sau:

% **telnet rama.poly.edu 79** Trying 128.238.10.212... Connected to rama.poly.edu. Escape character is '^]'.

Login Name

jacola Jane Colaginae

marcus Marcus Tullius

dialup11.poly.e

matewan Sepin Matewan

hengpi Heng Pin

nadats Nabeel Datsun

matewan Sepin Matewan

Connection closed by foreign host.

Or, requesting information about a specific user:

% **telnet rama.poly.edu 79** Trying 128.238.10.212... Connected to rama.poly.edu. Escape character is '^]'.**marcus**

Login Name

marcus Marcus Tullius

dialup11.poly.e

TTY Idle When Where

pts/15 13d Tue 17:33 farm-

TTY Idle When Where

\*pts/7 Tue 08:01 208.34.37.104

pts/15 13d Tue 17:33 farm-

\*pts/17 17: Thu 15:32 128.238.10.177

\*pts/10

pts/12

Tue 10:36 128.238.18.119

56 Mon 10:38 128.238.213.227

4 Sun 18:39 128.238.10.177

\*pts/8

Since there's no standard for the format of a finger URL, we will start by creating one. Ideally, this should look as much like an http URL as possible. Therefore, we will implement a finger URL like this:

Vì không có tiêu chuẩn nào cho định dạng của một URL ngón tay, chúng tôi sẽ bắt đầu bằng cách tạo một URL. Lý tưởng nhất là URL này càng giống URL http càng tốt. Do đó, chúng tôi sẽ triển khai một URL ngón tay như thế này:

finger://hostname:port/username

Second, we need to determine the content type returned by the finger protocol's getContentType( ) method. New protocols such as HTTP use MIME headers to indicate the content type; in these cases, you do not need to override the default getContentType( ) method provided by the URLConnection class. However, since most protocols precede MIME, you often need to specify the MIME type explicitly or use the static methods URLConnection.guess Content TypeFromName(String name) and URLConnection.guessContent TypeFromStream(InputStream in) to make an educated guess. This example doesn't need anything so complicated, however. A finger server returns ASCII text, so the getContentType( ) method should return the string text/plain. The text/plain MIME type has the advantage that Java already understands it. In the next chapter, you'll learn how to write content handlers that let Java understand additional MIME types.

Example 16.2 is a FingerURLConnection class that subclasses URLConnection. This class overrides the getContentType( ) and getInputStream( ) methods of URLConnection and implements connect( ). It also has a constructor that builds a new URLConnection from a URL.

Thứ hai, chúng ta cần xác định loại nội dung được trả về bởi phương thức getContentType () của giao thức ngón tay. Các giao thức mới như HTTP sử dụng tiêu đề MIME để chỉ ra loại nội dung; trong những trường hợp này, bạn không cần ghi đè phương thức getContentType () mặc định do lớp URLConnection cung cấp. Tuy nhiên, vì hầu hết các giao thức có trước MIME, bạn thường cần chỉ định kiểu MIME một cách rõ ràng hoặc sử dụng các phương thức tĩnh URLConnection.guess Content TypeFromName (String name) và URLConnection.guessContent TypeFromStream (InputStream in) để đưa ra một phỏng đoán thông minh. Tuy nhiên, ví dụ này không cần bất cứ điều gì quá phức tạp. Máy chủ ngón tay trả về văn bản ASCII, vì vậy phương thức getContentType () sẽ trả về chuỗi văn bản / đơn giản. Kiểu MIME văn bản / thuần túy có ưu điểm là Java đã hiểu nó. Trong chương tiếp theo, bạn sẽ học cách viết trình xử lý nội dung cho phép Java hiểu các kiểu MIME bổ sung.

Ví dụ 16.2 là một lớp FingerURLConnection phân lớp con URLConnection. Lớp này ghi đè các phương thức getContentType () và getInputStream () của URLConnection và thực hiện connect (). Nó cũng có một hàm tạo xây dựng một URLConnection mới từ một URL.

Example 16.2. The FingerURLConnection Class

package com.macfaq.net.www.protocol.finger;

import java.net.\*;

import java.io.\*;

public class FingerURLConnection extends URLConnection {

private Socket connection = null;

public final static int DEFAULT\_PORT = 79;

public FingerURLConnection(URL u) {

super(u); }

public synchronized InputStream getInputStream( ) throws

IOException {

if (!connected) this.connect( );

InputStream in = this.connection.getInputStream( );

return in;

}

public String getContentType( ) {

return "text/plain";

}

public synchronized void connect( ) throws IOException {

if (!connected) {

int port = url.getPort( );

if ( port < 1 || port > 65535) {

port = DEFAULT\_PORT;

}

this.connection = new Socket(url.getHost( ), port);

OutputStream out = this.connection.getOutputStream( );

String names = url.getFile( );

if (names != null && !names.equals("")) {

// delete initial /

names = names.substring(1);

names = URLDecoder.decode(names);

byte[] result;

try {

result = names.getBytes("ASCII");

}

catch (UnsupportedEncodingException e) {

result = names.getBytes( );

}

out.write(result);

}

out.write('\r');

out.write('\n');

out.flush( );

this.connected = true;

} }

}

This class has two fields. connection is a Socket between the client and the server. Both the getInputStream( ) method and the connect( ) method need access to this field, so it can't be a local variable. The second field is DEFAULT\_PORT, a final static int, that contains the finger protocol's default port; this port is used if the URL does not specify the port explicitly.

The class's constructor holds no surprises. It just calls the superclass's constructor with the same argument, the URL u. The connect( ) method opens a connection to the specified server on the specified port or, if no port is specified, then to the default finger port, 79. It sends the necessary request to the finger server. If any usernames were specified in the file part of the URL, they're sent. Otherwise, a blank line is sent. Assuming the connection is successfully opened (no exception is thrown), it sets the boolean field connected to true. Recall from the previous chapter that connected is a protected field in java.net.URLConnection, which is inherited by this subclass. The Socket that connect( ) opens is stored in the field connection for later use by getInputStream( ). The connect( ) and getInputStream( ) methods are synchronized to avoid a possible race condition on the connected variable.

The getContentType( ) method returns a String containing a MIME type for the data. This is used by the getContent( ) method of java.net.URLConnection to select the appropriate content handler. The data returned by a finger server is almost always ASCII text or some reasonable approximation thereof, so this getContentType( ) method always returns text/plain. The getInputStream( ) method returns an InputStream, which it gets from the Socket that connect created. If the connection has not already been established when getInputStream( ) is called, the method calls connect( ) itself.

Once you have a URLConnection, you need a subclass of URLStreamHandler that knows how to handle a finger server. This class needs an openConnection( ) method that builds a new FingerURLConnection from a URL. Since we defined the finger URL so that it is similar to an http URL, we don't need to implement a parseURL( ) method. Example 16.3 is a stream handler for the finger protocol. For the moment, we're going to use Sun's convention for naming protocol handlers, so we call this class Handler and place it in the package com.macfaq.net.[www.protocol.finger](http://www.protocol.finger/).

Lớp này có hai trường. kết nối là một Socket giữa máy khách và máy chủ. Cả phương thức getInputStream () và phương thức connect () đều cần quyền truy cập vào trường này, vì vậy nó không thể là một biến cục bộ. Trường thứ hai là DEFAULT\_PORT, một int tĩnh cuối cùng, chứa cổng mặc định của giao thức ngón tay; cổng này được sử dụng nếu URL không chỉ định cổng một cách rõ ràng. Hàm tạo của lớp không có gì đáng ngạc nhiên. Nó chỉ gọi hàm tạo của lớp cha với cùng một đối số, URL u. Phương thức connect () mở một kết nối đến máy chủ được chỉ định trên cổng được chỉ định hoặc nếu không có cổng nào được chỉ định, thì đến cổng ngón tay mặc định, 79. Nó sẽ gửi yêu cầu cần thiết đến máy chủ ngón tay. Nếu bất kỳ tên người dùng nào được chỉ định trong phần tệp của URL, chúng sẽ được gửi đi. Nếu không, một dòng trống sẽ được gửi. Giả sử kết nối được mở thành công (không có ngoại lệ nào được đưa ra), nó sẽ đặt trường boolean được kết nối thành true. Nhớ lại chương trước rằng kết nối là một trường được bảo vệ trong java.net.URLConnection, được kế thừa bởi lớp con này. Socket kết nối () mở ra được lưu trữ trong kết nối trường để getInputStream () sử dụng sau này. Các phương thức connect () và getInputStream () được đồng bộ hóa để tránh tình trạng chạy đua có thể xảy ra trên biến được kết nối. Phương thức getContentType () trả về một Chuỗi chứa kiểu MIME cho dữ liệu. Điều này được sử dụng bởi phương thức getContent () của java.net.URLConnection để chọn trình xử lý nội dung thích hợp. Dữ liệu do máy chủ ngón tay trả về hầu như luôn luôn là văn bản ASCII hoặc một số ước lượng hợp lý của chúng, vì vậy phương thức getContentType () này luôn trả về văn bản / đơn giản. Phương thức getInputStream () trả về một InputStream, nó nhận được từ Socket kết nối đã tạo. Nếu kết nối chưa được thiết lập khi getInputStream () được gọi, thì chính phương thức gọi connect (). Khi bạn có URLConnection, bạn cần một lớp con của URLStreamHandler biết cách xử lý một máy chủ ngón tay. Lớp này cần một phương thức openConnection () để tạo một FingerURLConnection mới từ một URL. Vì chúng tôi đã xác định URL ngón tay để nó giống với URL http, nên chúng tôi không cần triển khai phương thức parseURL ().

Ví dụ 16.3 là một trình xử lý luồng cho giao thức ngón tay. Hiện tại, chúng tôi sẽ sử dụng quy ước của Sun để đặt tên cho các trình xử lý giao thức, vì vậy chúng tôi gọi lớp này là Handler và đặt nó trong gói com.macfaq.net.www.protocol.finger.

Example 16.3. The Finger Handler Class

package com.macfaq.net.www.protocol.finger;

import java.net.\*;

import java.io.\*;

public class Handler extends URLStreamHandler {

public int getDefaultPort( ) {

return 79; }

protected URLConnection openConnection(URL u) throws IOException {

return new FingerURLConnection(u);

}

}

You can use HotJava to test this protocol handler. Add the following line to

your .hotjava/properties file or some other place from which HotJava will load it:

Bạn có thể sử dụng HotJava để kiểm tra trình xử lý giao thức này. Thêm dòng sau vào tệp .hotjava / thuộc tính của bạn hoặc một số nơi khác mà từ đó HotJava sẽ tải nó:

java.protocol.handler.pkgs=com.macfaq.net.www.protocol

Some (but not all) versions of HotJava may also allow you to set the property from the command line:

Một số (nhưng không phải tất cả) phiên bản HotJava cũng có thể cho phép bạn đặt thuộc tính từ dòng lệnh:

% **hotjava -Djava.protocol.handler.pkgs=com.macfaq.net.www.protocol**

You also need to make sure that your classes are somewhere in HotJava's class path. Note that HotJava does not normally use the CLASSPATH environment variable to look for classes, so just putting them someplace where the JDK or JRE can find them may not be sufficient. Using HotJava 3.0 on Windows with the JDK 1.3b1, I was able to put my classes in the jdk1.3/jre/lib/classes folder. Your mileage may vary depending on what version of HotJava you're using with which version of the JDK on which platform.

Run it, and ask for a URL of a site running finger, such as utopia.poly.edu. Figure 16.1 shows the result.

Bạn cũng cần đảm bảo rằng các lớp của bạn ở đâu đó trong đường dẫn lớp của HotJava. Lưu ý rằng HotJava thường không sử dụng biến môi trường CLASSPATH để tìm kiếm các lớp, vì vậy chỉ cần đặt chúng ở một nơi nào đó mà JDK hoặc JRE có thể tìm thấy chúng có thể không đủ. Sử dụng HotJava 3.0 trên Windows với JDK 1.3b1, tôi đã có thể đặt các lớp của mình vào thư mục jdk1.3 / jre / lib / class. Số dặm của bạn có thể thay đổi tùy thuộc vào phiên bản HotJava bạn đang sử dụng với phiên bản JDK trên nền tảng nào.

Chạy nó và yêu cầu URL của một trang web đang chạy, chẳng hạn như utopia.poly.edu. Hình 16.1 cho thấy kết quả.

Figure 16.1. HotJava using the finger protocol handler

16.4 More Protocol Handler Examples and Techniques - Ví dụ và kỹ cách kỹ thuật xử lý giao thức

Now that you've seen how to write one protocol handler, it's not at all difficult to write more. Remember the five basic steps of creating a new protocol handler:

1. Design a URL for the protocol, if a standard URL for that protocol doesn't already exist. As of July 2000, the official list of URL schemes at the IANA (http://www.isi.edu/in-notes/iana/assignments/url-schemes) includes only 29 different schemes and reserves three more. For anything else, you need to define your own. Make your new URL as similar to an http URL as possible.
2. Decide what MIME type should be returned by the protocol handler's getContentType( ) method. The text/plain content type is often appropriate for legacy protocols. Another option is to convert the incoming data to HTML inside getInputStream( ) and return text/html. Binary data often uses one of

the many application types. In some cases, you may be able to use the URLConnection.guessContentTypeFromName( ) or URLConnection.guessContentTypeFromStream( ) methods to determine the right MIME type.

1. Write a subclass of URLConnection that understands this protocol. It should implement the connect( ) method and may override the getContent Type( ), getOutputStream( ), and getInputStream( ) methods of URL Connection. It also needs a constructor that builds a new URLConnection from a URL.
2. Write a subclass of URLStreamHandler with an openConnection( ) method that knows how to return a new instance of your subclass of URLConnection. Also provide a getDefaultPort( ) method that returns the well-known port for the protocol. If your URL does not look like an http URL, override parseURL( ) and toExternalForm( ) as well.
3. Implement the URLStreamHandlerFactory interface and the createStream Handler( ) method in a convenient class.

Let's look at handlers for two more protocols, daytime and chargen, that will bring up different challenges.

Bây giờ bạn đã thấy cách viết một trình xử lý giao thức, không khó để viết thêm. Hãy nhớ năm bước cơ bản để tạo một trình xử lý giao thức mới: 1. Thiết kế một URL cho giao thức, nếu một URL chuẩn cho giao thức đó chưa tồn tại. Kể từ tháng 7 năm 2000, danh sách chính thức của các lược đồ URL tại IANA (http://www.isi.edu/in-notes/iana/assignments/url-schemes) chỉ bao gồm 29 lược đồ khác nhau và dự trữ thêm ba lược đồ khác. Đối với bất cứ điều gì khác, bạn cần phải xác định của riêng bạn. Tạo URL mới của bạn giống với URL http nhất có thể. 2. Quyết định kiểu MIME nào sẽ được trả về bằng phương thức getContentType () của trình xử lý giao thức. Loại nội dung văn bản / nội dung thuần túy thường thích hợp cho các giao thức kế thừa. Một tùy chọn khác là chuyển đổi dữ liệu đến sang HTML bên trong getInputStream () và trả về văn bản / html. Dữ liệu nhị phân thường sử dụng một trong các nhiều loại ứng dụng. Trong một số trường hợp, bạn có thể sử dụng phương thức URLConnection.guessContentTypeFromName () hoặc URLConnection.guessContentTypeFromStream () để xác định loại MIME phù hợp. 3. Viết một lớp con của URLConnection hiểu giao thức này. Nó phải triển khai phương thức connect () và có thể ghi đè các phương thức getContent Type (), getOutputStream () và getInputStream () của URL Connection. Nó cũng cần một hàm tạo xây dựng một URLConnection mới từ một URL. 4. Viết một lớp con của URLStreamHandler với một phương thức openConnection () biết cách trả về một phiên bản mới của lớp con URLConnection của bạn. Đồng thời cung cấp phương thức getDefaultPort () trả về cổng nổi tiếng cho giao thức. Nếu URL của bạn không giống URL http, hãy ghi đè parseURL () và toExternalForm (). 5. Triển khai giao diện URLStreamHandlerFactory và phương thức createStream Handler () trong một lớp thuận tiện. Chúng ta hãy xem xét các trình xử lý cho hai giao thức khác, ban ngày và chargen, sẽ đưa ra những thách thức khác nhau.

16.4.1 A daytime Protocol Handler - Trình xử lý giao thức ban ngày

For a daytime protocol handler, let's say that the URL should look like daytime://metalab.unc.edu. We'll allow for nonstandard port assignments in the same way as with HTTP: follow the hostname with a colon and the port (daytime://metalab.unc.edu:2082). Finally, allow a terminating slash, and ignore everything following the slash. For example, daytime://metalab.unc.edu/index.html is equivalent to daytime://metalab.unc.edu. This is similar enough to an http URL that you'll be able to use the default toExternalForm( ) and parseURL( ) methods.

Although the content returned by the daytime protocol is really text/plain, this protocol handler is going to reformat the data into an HTML page. Then it can return a content type of text/html and let the web browser display it more dramatically. The resulting HTML will look like this:

Đối với trình xử lý giao thức ban ngày, giả sử rằng URL sẽ giống như ban ngày: //metalab.unc.edu. Chúng tôi sẽ cho phép gán cổng không chuẩn theo cách tương tự như với HTTP: theo sau tên máy chủ bằng dấu hai chấm và cổng (daily: //metalab.unc.edu: 2082). Cuối cùng, cho phép một dấu gạch chéo kết thúc và bỏ qua mọi thứ sau dấu gạch chéo. Ví dụ: ban ngày: //metalab.unc.edu/index.html tương đương với ban ngày: //metalab.unc.edu. Điều này đủ tương tự với một URL http mà bạn sẽ có thể sử dụng các phương thức toExternalForm () và parseURL () mặc định. Mặc dù nội dung được trả về bởi giao thức ban ngày thực sự là văn bản / đơn giản, trình xử lý giao thức này sẽ định dạng lại dữ liệu thành một trang HTML. Sau đó, nó có thể trả về một loại nội dung là văn bản / html và cho phép trình duyệt web hiển thị nó một cách rõ ràng hơn. HTML kết quả sẽ giống như sau:

<html><head><title>The Time at metalab.unc.edu</title></head><body> <h1>Fri Oct 29 14:32:07 1999</h1>

</body></html>

The trick is that the page can be broken up into three different strings:

* Everything before the time
* The time
* Everything after the time  
  The first and the third strings can be calculated before the connection is even opened. We'll formulate these as byte arrays of ASCII text and use them to create two ByteArrayInputStreams. Then we'll use a SequenceInputStream to combine those two streams with the data actually returned from the server. Example 16.4 demonstrates. This is a neat trick for protocols such as daytime that return a very

limited amount of data; it can be inserted in a single place in an HTML document. Protocols such as finger that return more complex and less predictable text might need to use a FilterInputStream that inserts the HTML on the fly instead. And of course a third possibility is simply to return a custom content type and use a custom content handler to display it. This third option will be explored in the next chapter.

Bí quyết là trang có thể được chia thành ba chuỗi khác nhau: • Mọi thứ trước thời điểm • Thời gian • Mọi thứ sau thời gian Chuỗi đầu tiên và chuỗi thứ ba có thể được tính toán trước khi kết nối được mở. Chúng tôi sẽ định dạng chúng dưới dạng mảng byte của văn bản ASCII và sử dụng chúng để tạo hai ByteArrayInputStreams. Sau đó, chúng tôi sẽ sử dụng SequenceInputStream để kết hợp hai luồng đó với dữ liệu thực sự được trả về từ máy chủ. Ví dụ 16.4 chứng minh. Đây là một mẹo nhỏ cho các giao thức chẳng hạn như ban ngày trả lại rất số lượng dữ liệu hạn chế; nó có thể được chèn vào một nơi duy nhất trong tài liệu HTML. Các giao thức như ngón tay trả về văn bản phức tạp hơn và ít dễ đoán hơn có thể cần sử dụng FilterInputStream để chèn HTML nhanh chóng. Và tất nhiên khả năng thứ ba chỉ đơn giản là trả về một loại nội dung tùy chỉnh và sử dụng trình xử lý nội dung tùy chỉnh để hiển thị nó. Tùy chọn thứ ba này sẽ được khám phá trong chương tiếp theo.

Example 16.4. The DaytimeURLConnection Class

package com.macfaq.net.www.protocol.daytime;

import java.net.\*;

import java.io.\*;

public class DaytimeURLConnection extends URLConnection {

private Socket connection = null;

public final static int DEFAULT\_PORT = 13;

public DaytimeURLConnection (URL u) {

super(u); }

public synchronized InputStream getInputStream( ) throws

IOException {

if (!connected) connect( );

String header = "<html><head><title>The Time at "

+ url.getHost( ) + "</title></head><body><h1>";

String footer = "</h1></body></html>";

InputStream in1 = new

ByteArrayInputStream(header.getBytes("8859\_1"));

InputStream in2 = this.connection.getInputStream( );

InputStream in3 = new

ByteArrayInputStream(footer.getBytes("8859\_1"));

SequenceInputStream result = new SequenceInputStream(in1, in2);

result = new SequenceInputStream(result, in3);

return result;

}

public String getContentType( ) {

return "text/html";

}

public synchronized void connect( ) throws IOException {

if (!connected) {

int port = url.getPort( );

if ( port <= 0 || port > 65535) {

port = DEFAULT\_PORT;

}

this.connection = new Socket(url.getHost( ), port);

this.connected = true;

}

} }

This class declares two fields. The first is connection, which is a Socket between the client and the server. The second field is DEFAULT\_PORT, a final static int variable, that holds the default port for the daytime protocol (port 13) and is used if the URL doesn't specify the port explicitly.

The constructor has no surprises. It just calls the superclass's constructor with the same argument, the URL u. The connect( ) method opens a connection to the specified server on the specified port (or, if no port is specified, to the default port); if the connection is opened successfully, connect( ) sets the boolean variable connected to true. Recall from the previous chapter that connected is a protected field in URLConnection that is inherited by this subclass. The Socket that's opened by this method is stored in the connection field for later use by getInputStream( ).

The getContentType( ) method returns a String containing a MIME type for the data. This method is called by the getContent( ) method of URLConnection to select the appropriate content handler. The getInputStream( ) method reformats the text into HTML, so the getContentType( ) method returns text/html.

The getInputStream( ) method builds a SequenceInputStream out of several string literals, the host property of url, and the actual stream provided by the Socket connecting the client to the server. If the socket is not connected when this method is called, then the method calls connect( ) to establish the connection.

Next, you need a subclass of URLStreamHandler that knows how to handle a daytime server. This class needs an openConnection( ) method that builds a new DaytimeURLConnection from a URL and a getDefaultPort( ) method that returns the well-known daytime port 13. Since the daytime URL has been made similar to an http URL, we don't need to override parseURL( ); once we have written openConnection( ), we're done. Example 16.5 shows the daytime protocol's URLStreamHandler.

Lớp này khai báo hai trường. Đầu tiên là kết nối, là một Socket giữa máy khách và máy chủ. Trường thứ hai là DEFAULT\_PORT, một biến int tĩnh cuối cùng, giữ cổng mặc định cho giao thức ban ngày (cổng 13) và được sử dụng nếu URL không chỉ định cổng rõ ràng. Người xây dựng không có bất ngờ. Nó chỉ gọi hàm tạo của lớp cha với cùng một đối số, URL u. Phương thức connect () mở một kết nối đến máy chủ được chỉ định trên cổng được chỉ định (hoặc, nếu không có cổng nào được chỉ định, đến cổng mặc định); nếu kết nối được mở thành công, connect () đặt biến boolean được kết nối thành true. Nhớ lại chương trước rằng kết nối là một trường được bảo vệ trong URLConnection được kế thừa bởi lớp con này. Socket được mở bằng phương pháp này được lưu trữ trong trường kết nối để getInputStream () sử dụng sau này. Phương thức getContentType () trả về một Chuỗi chứa kiểu MIME cho dữ liệu. Phương thức này được gọi bởi phương thức getContent () của URLConnection để chọn trình xử lý nội dung thích hợp. Phương thức getInputStream () định dạng lại văn bản thành HTML, vì vậy phương thức getContentType () trả về văn bản / html. Phương thức getInputStream () xây dựng một SequenceInputStream từ một số ký tự chuỗi, thuộc tính máy chủ của url và luồng thực được cung cấp bởi Socket kết nối máy khách với máy chủ. Nếu ổ cắm không được kết nối khi phương thức này được gọi, thì phương thức gọi connect () để thiết lập kết nối. Tiếp theo, bạn cần một lớp con của URLStreamHandler biết cách xử lý một máy chủ ban ngày. Lớp này cần một phương thức openConnection () để xây dựng một DaytimeURLConnection mới từ một URL và một phương thức getDefaultPort () trả về cổng ban ngày nổi tiếng 13. Vì URL ban ngày đã được tạo tương tự như một URL http, chúng tôi không cần để ghi đè parseURL (); khi chúng ta đã viết openConnection (), chúng ta đã hoàn tất. Ví dụ 16.5 cho thấy URLStreamHandler của giao thức ban ngày.

Example 16.5. The DaytimeURLStreamHandler Class

package com.macfaq.net.www.protocol.daytime;

import java.net.\*;

import java.io.\*;

public class Handler extends URLStreamHandler {

public int getDefaultPort( ) {

return 13; }

protected URLConnection openConnection(URL u) throws IOException {

return new DaytimeURLConnection(u);

}

}

Since we've used the same package-naming convention here as for the previous finger protocol handler, no further changes to HotJava's properties need to be made to let HotJava find this. Just compile the files, put the classes somewhere in HotJava's class

path, and load a URL that points to an active daytime server. Figure 16.2 demonstrates.

Vì chúng tôi đã sử dụng quy ước đặt tên gói ở đây giống như đối với trình xử lý giao thức ngón tay trước đó, nên không cần thực hiện thêm thay đổi nào đối với các thuộc tính của HotJava để cho phép HotJava tìm thấy điều này. Chỉ cần biên dịch các tệp, đặt các lớp ở đâu đó trong lớp của HotJava và tải một URL trỏ đến một máy chủ hoạt động ban ngày. Hình 16.2 minh họa.

Figure 16.2. HotJava using our daytime protocol handler

16.4.2 A chargen Protocol Handler - Trình xử lý giao thức hàng loạt

The chargen protocol, which is defined in RFC 864, is a very simple protocol

designed for testing clients. The server listens for connections on port 19. When a client connects, the server sends an endless stream of characters until the client disconnects. Any input from the client is ignored. The RFC does not specify which character sequence to send, but recommends that the server use a recognizable pattern. One common pattern is rotating, 72-character carriage return/linefeed delimited lines of the 95 ASCII printing characters like this:

Giao thức chargen, được định nghĩa trong RFC 864, là một giao thức rất đơn giản được thiết kế cho khách hàng thử nghiệm. Máy chủ lắng nghe các kết nối trên cổng 19. Khi một máy khách kết nối, máy chủ sẽ gửi một luồng ký tự vô tận cho đến khi máy khách ngắt kết nối. Bất kỳ đầu vào nào từ máy khách đều bị bỏ qua. RFC không chỉ định chuỗi ký tự nào sẽ gửi, nhưng khuyến nghị rằng máy chủ sử dụng một mẫu dễ nhận biết. Một mẫu phổ biến là xoay, các dòng phân cách 72 ký tự xuống dòng / dòng phân cách dòng của 95 ký tự in ASCII như thế này:

!"#$%&'( )\*+,- ./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^\_`abcdefgh "#$%&'( )\*+,- ./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^\_`abcdefghi #$%&'( )\*+,- ./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^\_`abcdefghij $%&'( )\*+,- ./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^\_`abcdefghijk %&'( )\*+,- ./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^\_`abcdefghijkl &'( )\*+,- ./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^\_`abcdefghijklm '( )\*+,- ./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^\_`abcdefghijklmn ( )\*+,- ./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^\_`abcdefghijklmno

The big trick with this protocol is deciding when to stop. A TCP chargen server sends an unlimited amount of data. Most web browsers don't deal well with this. HotJava won't even attempt to display a file until the end of stream is seen. Consequently, the first thing we'll need is a FilterInputStream subclass that cuts off the server (or at least starts ignoring it) after a certain amount of data has been sent. Example 16.6 is such a class.

Bí quyết lớn với giao thức này là quyết định khi nào thì dừng lại. Máy chủ ký tự TCP gửi một lượng dữ liệu không giới hạn. Hầu hết các trình duyệt web không giải quyết tốt vấn đề này. HotJava thậm chí sẽ không cố gắng hiển thị một tệp cho đến khi nhìn thấy phần cuối của luồng. Do đó, điều đầu tiên chúng ta cần là một lớp con FilterInputStream cắt máy chủ (hoặc ít nhất là bắt đầu bỏ qua nó) sau khi một lượng dữ liệu nhất định đã được gửi đi. Ví dụ 16.6 là một lớp như vậy.

Example 16.6. FiniteInputStream

package com.macfaq.io;

import java.io.\*;

public class FiniteInputStream extends FilterInputStream {

private int limit = 8192;

private int bytesRead = 0;

public FiniteInputStream(InputStream in) {

this(in, 8192);

}

public FiniteInputStream(InputStream in, int limit) {

super(in);

this.limit = limit;

}

public int read( ) throws IOException {

if (bytesRead >= limit) return -1;

int c = in.read( );

bytesRead++;

return c;

}

public int read(byte[] data) throws IOException {

return this.read(data, 0, data.length);

}

public int read(byte[] data, int offset, int length)

throws IOException {

if (data == null) throw new NullPointerException( );

else if ((offset < 0) || (offset > data.length) || (length <

0) ||

{

((offset + length) > data.length) || ((offset + length) < 0))

throw new IndexOutOfBoundsException( );

}

else if (length == 0) {

return 0; }

if (bytesRead >= limit) return -1;

else if (bytesRead + length > limit) {

int numToRead = bytesRead + length - limit;

int numRead = in.read(data, offset, numToRead);

if (numRead == -1) return -1;

bytesRead += numRead;

return numRead;

}

else { // will not exceed limit

int numRead = in.read(data, offset, length);

if (numRead == -1) return -1;

bytesRead += numRead;

return numRead;

} }

public int available( ) throws IOException {

if (bytesRead >= limit) return 1;

else return in.available( );

}

}

Next, since there's no standard for the format of a chargen URL, we have to create one. Ideally, this should look as much like an http URL as possible. Therefore, we will implement a chargen URL like this:

Tiếp theo, vì không có tiêu chuẩn nào cho định dạng của một URL ký tự, nên chúng tôi phải tạo một định dạng. Lý tưởng nhất là URL này càng giống URL http càng tốt. Do đó, chúng tôi sẽ triển khai một URL ký tự như thế này:

chargen://hostname:port

Second, we need to choose the content type to be returned by the chargen protocol's getContentType( ) method. A chargen server returns ASCII text, so the getContentType( ) method should return the string text/plain. The text/plain MIME type has the advantage that Java already understands it.

Example 16.7 is a ChargenURLConnection class that subclasses URLConnection. This class overrides the getContentType( ) and getInputStream( ) methods of URLConnection and implements connect( ). It also has a constructor that builds a new URLConnection from a URL.

Thứ hai, chúng ta cần chọn loại nội dung được trả về bởi phương thức getContentType () của giao thức chargen. Máy chủ chargen trả về văn bản ASCII, vì vậy phương thức getContentType () sẽ trả về chuỗi văn bản / đơn giản. Kiểu MIME văn bản / thuần túy có ưu điểm là Java đã hiểu nó. Ví dụ 16.7 là một lớp ChargenURLConnection phân lớp URLConnection. Lớp này ghi đè các phương thức getContentType () và getInputStream () của URLConnection và thực hiện connect (). Nó cũng có một hàm tạo xây dựng một URLConnection mới từ một URL.

Example 16.7. The ChargenURLConnection Class

package com.macfaq.net.www.protocol.chargen;

import java.net.\*;

import java.io.\*;

import com.macfaq.io.\*;

public class ChargenURLConnection extends URLConnection {

private Socket connection = null;

public final static int DEFAULT\_PORT = 19;

public ChargenURLConnection(URL u) {

super(u); }

public synchronized InputStream getInputStream( ) throws

IOException {

if (!connected) this.connect( );

return new FiniteInputStream(this.connection.getInputStream( ));

}

public String getContentType( ) {

return "text/plain";

}

public synchronized void connect( ) throws IOException {

if (!connected) {

int port = url.getPort( );

if ( port < 1 || port > 65535) {

port = DEFAULT\_PORT;

}

this.connection = new Socket(url.getHost( ), port);

this.connected = true;

} }

}

This class has two fields. connection is a Socket between the client and the server. The second field is DEFAULT\_PORT, a final static int that contains the chargen protocol's default port; this port is used if the URL does not specify the port explicitly.

The class's constructor just passes the URL u to the superclass's constructor. The connect( ) method opens a connection to the specified server on the specified port (or, if no port is specified, then to the default chargen port, 19) and, assuming the connection is successfully opened, sets the boolean field connected to true. The Socket that connect( ) opens is stored in the field connection for later use by getInputStream( ). The connect( ) method is synchronized to avoid a possible race condition on the connected variable.

The getContentType( ) method returns a String containing a MIME type for the data. The data returned by a chargen server is always ASCII text, so this getContentType( ) method always returns text/plain.

The getInputStream( ) connects if necessary, then gets the InputStream from this.connection. Rather than returning it immediately, getInputStream( ) first chains it to a FiniteInputStream.

Now that we have a URLConnection, we need a subclass of URLStreamHandler that knows how to handle a chargen server. This class needs an openConnection( ) method that builds a new ChargenURLConnection from a URL and agetDefaultPort( ) method that returns the well-known chargen port. Since we defined the chargen URL so that it is similar to an http URL, we don't need to implement a parseURL( ) method. Example 16.8 is a stream handler for the chargen protocol.

Lớp này có hai trường. kết nối là một Socket giữa máy khách và máy chủ. Trường thứ hai là DEFAULT\_PORT, một int tĩnh cuối cùng chứa cổng mặc định của giao thức chargen; cổng này được sử dụng nếu URL không chỉ định cổng một cách rõ ràng. Phương thức khởi tạo của lớp chỉ chuyển URL u đến phương thức khởi tạo của lớp cha. Phương thức connect () mở kết nối đến máy chủ được chỉ định trên cổng được chỉ định (hoặc, nếu không có cổng nào được chỉ định, thì đến cổng chargen mặc định, 19) và, giả sử kết nối được mở thành công, hãy đặt trường boolean được kết nối thành true . Socket kết nối () mở ra được lưu trữ trong kết nối trường để getInputStream () sử dụng sau này. Phương thức connect () được đồng bộ hóa để tránh điều kiện chạy đua có thể xảy ra trên biến được kết nối. Phương thức getContentType () trả về một Chuỗi chứa kiểu MIME cho dữ liệu. Dữ liệu do máy chủ chargen trả về luôn là văn bản ASCII, vì vậy phương thức getContentType () này luôn trả về văn bản / thuần. GetInputStream () kết nối nếu cần, sau đó lấy InputStream từ this.connection. Thay vì trả lại nó ngay lập tức, getInputStream () đầu tiên chuỗi nó thành FiniteInputStream. Bây giờ chúng ta đã có URLConnection, chúng ta cần một lớp con của URLStreamHandler biết cách xử lý một máy chủ chargen. Lớp này cần một phương thức openConnection () để xây dựng một ChargenURLConnection mới từ một URL và phương thức agetDefaultPort () trả về cổng chargen nổi tiếng. Vì chúng tôi đã xác định URL chargen để nó giống với URL http, nên chúng tôi không cần triển khai phương thức parseURL (). Ví dụ 16.8 là một trình xử lý luồng cho giao thức chargen.

Example 16.8. The chargen Handler Class

package com.macfaq.net.www.protocol.chargen;

import java.net.\*;

import java.io.\*;

public class Handler extends URLStreamHandler {

public int getDefaultPort( ) {

return 19; }

protected URLConnection openConnection(URL u) throws IOException {

return new ChargenURLConnection(u);

}

}

You can use HotJava to test this protocol handler. Run it, and ask for a URL of a site running a chargen server, such as vision.poly.edu. Figure 16.3 shows the result.

Bạn có thể sử dụng HotJava để kiểm tra trình xử lý giao thức này. Chạy nó và yêu cầu URL của một trang web chạy máy chủ chargen, chẳng hạn như vision.poly.edu. Hình 16.3 cho thấy kết quả.

Figure 16.3. HotJava using the chargen protocol handler

16.5 The URLStreamHandlerFactory Interface

The last section showed you how to install new protocol handlers that you wrote into HotJava, an application that someone else wrote. However, if you write your own application, you can implement your own scheme for finding and load ing protocol handlers. The easiest way to do this is to install a URLStream HandlerFactory in your application:

Phần cuối cùng đã hướng dẫn bạn cách cài đặt các trình xử lý giao thức mới mà bạn đã viết vào HotJava, một ứng dụng mà người khác đã viết. Tuy nhiên, nếu bạn viết ứng dụng của riêng mình, bạn có thể triển khai lược đồ của riêng mình để tìm và tải các trình xử lý giao thức ing. Cách dễ nhất để làm điều này là cài đặt URLStream HandlerFactory trong ứng dụng của bạn:

public abstract interface URLStreamHandlerFactory

Only applications are allowed to install a new URL StreamHandlerFactory. Applets that run in the applet viewer or a web browser must use the URLStreamHandlerFactory that is provided. An attempt to set a different one will fail, either because another factory is already installed or because of a Security Exception.

Chỉ các ứng dụng mới được phép cài đặt URL StreamHandlerFactory mới. Các applet chạy trong trình xem applet hoặc trình duyệt web phải sử dụng URLStreamHandlerFactory được cung cấp. Nỗ lực đặt một cái khác sẽ không thành công, vì một nhà máy khác đã được cài đặt hoặc do một SecurityException.

The URLStreamHandlerFactory interface declares a single method, createURLStreamHandler( ):

public abstract URLStreamHandler createURLStreamHandler(String

protocol)

This method loads the appropriate protocol handler for the specified protocol. To use this method, write a class that implements the URLStreamHandlerFactory interface and include a createURLStreamHandler( ) method in that class. This method needs to know how to find the protocol handler for a given protocol. This is no more complicated than knowing the names and packages of the custom protocols you've implemented.

The createURLStreamHandler( ) method does not need to know the names of all

the installed protocol handlers. If it doesn't recognize a protocol, then it should simply return null. This tells Java to follow the default procedure for locating stream handlers; that is, to look for a class named *protocol*.Handler in one of the packages listed in the java.protocol.handler.pkgs system property or in sun.net.www.protocol.

To install the stream handler factory, pass an instance of the class that implements the URLStreamHandlerFactory interface to the static method URL.setURLStreamHandlerFactory( ) at the start of your program. Example 16.9 is a URLStreamHandlerFactory( ) whose createURLStreamHandler( ) method re cognizes the finger, daytime, and chargen protocols and returns the appropriate handler from the last several examples. Since these classes are all named Handler, fully package-qualified names are used.

Phương thức này tải trình xử lý giao thức thích hợp cho giao thức được chỉ định. Để sử dụng phương thức này, hãy viết một lớp triển khai giao diện URLStreamHandlerFactory và bao gồm một phương thức createURLStreamHandler () trong lớp đó. Phương pháp này cần biết cách tìm trình xử lý giao thức cho một giao thức nhất định. Điều này không phức tạp hơn việc biết tên và gói của các giao thức tùy chỉnh mà bạn đã triển khai. Phương thức createURLStreamHandler () không cần biết tên của tất cả các trình xử lý giao thức đã cài đặt. Nếu nó không nhận ra một giao thức, thì nó sẽ chỉ trả về null. Điều này yêu cầu Java tuân theo quy trình mặc định để định vị trình xử lý luồng; nghĩa là, để tìm kiếm một lớp có tên là protocol.Handler trong một trong các gói được liệt kê trong thuộc tính hệ thống java.protocol.handler.pkgs hoặc trong sun.net.www.protocol. Để cài đặt nhà máy xử lý luồng, hãy chuyển một phiên bản của lớp triển khai giao diện URLStreamHandlerFactory tới phương thức tĩnh URL.setURLStreamHandlerFactory () khi bắt đầu chương trình của bạn. Ví dụ 16.9 là một URLStreamHandlerFactory () có phương thức createURLStreamHandler () nhận dạng các giao thức ngón tay, ban ngày và chargen và trả về trình xử lý thích hợp từ một số ví dụ cuối cùng. Vì tất cả các lớp này đều được đặt tên là Handler, các tên đủ điều kiện gói hoàn toàn được sử dụng.

Example 16.9. A URLStreamHandlerFactory for finger, daytime, and chargen

package com.macfaq.net.www.protocol;

import java.net.\*;

public class NewFactory implements URLStreamHandlerFactory {

public URLStreamHandler createURLStreamHandler(String protocol) {

if (protocol.equalsIgnoreCase("finger")) {

return new com.macfaq.net.www.protocol.finger.Handler( );

}

else if (protocol.equalsIgnoreCase("chargen")) {

return new com.macfaq.net.www.protocol.chargen.Handler( );

}

else if (protocol.equalsIgnoreCase("daytime")) {

return new com.macfaq.net.www.protocol.daytime.Handler( );

}

else {

return null;

}

} }

We use the equalsIgnoreCase( ) method from java.lang.String to test the identity of the protocol because it shouldn't make a difference whether you ask for finger://rama.poly.edu or FINGER://RAMA.POLY.EDU. If the protocol is recognized, then createURLStreamHandler( ) creates an instance of the proper Handler class and returns it; otherwise, the method returns null, which tells the URL class to look for a URLStreamHandler in the standard locations.

Since browsers, HotJava included, generally don't allow you to install your own URLStreamHandlerFactory, this will be of use only in applications. Example 16.10 is a simple character mode program that uses this factory and its associated protocol handlers to print server data on System.out. Notice that it does not importcom.macfaq.net.www.protocol.chargen, com.macfaq.net.www.protocol.finger, orcom.macfaq.net.www.protocol.daytime. All this program knows is that it has a URL. It does not need to know how that protocol is handled or even how the right URLConnection object is instantiated.

Chúng tôi sử dụng phương thức equalsIgnoreCase () từ java.lang.String để kiểm tra danh tính của giao thức vì nó sẽ không tạo ra sự khác biệt cho dù bạn yêu cầu ngón tay: //rama.poly.edu hay FINGER: //RAMA.POLY. EDU. Nếu giao thức được nhận dạng, thì createURLStreamHandler () tạo một thể hiện của lớp Xử lý thích hợp và trả về nó; nếu không, phương thức này trả về null, điều này cho lớp URL tìm kiếm URLStreamHandler ở các vị trí tiêu chuẩn. Vì các trình duyệt, bao gồm HotJava, thường không cho phép bạn cài đặt URLStreamHandlerFactory của riêng mình, điều này sẽ chỉ được sử dụng trong các ứng dụng. Ví dụ 16.10 là một chương trình chế độ ký tự đơn giản sử dụng nhà máy này và các trình xử lý giao thức liên quan của nó để in dữ liệu máy chủ trên System.out. Lưu ý rằng nó không importcom.macfaq.net.www.protocol.chargen, com.macfaq.net.www.protocol.finger, orcom.macfaq.net.www.protocol.daytime. Tất cả những gì chương trình này biết là nó có một URL. Nó không cần biết giao thức đó được xử lý như thế nào hoặc thậm chí cách khởi tạo đối tượng URLConnection phù hợp.

Example 16.10. A SourceViewer Program That Sets a URLStreamHandlerFactory

import java.net.\*;

import java.io.\*;

import com.macfaq.net.www.protocol.\*;

public class SourceViewer3 {

public static void main (String[] args) {

URL.setURLStreamHandlerFactory(new NewFactory( ));

if (args.length > 0) {

try {

//Open the URL for reading

URL u = new URL(args[0]);

InputStream in = new BufferedInputStream(u.openStream( ));

// chain the InputStream to a Reader

Reader r = new InputStreamReader(in);

int c;

while ((c = r.read( )) != -1) {

System.out.print((char) c);

}

}

catch (MalformedURLException e) {

System.err.println(args[0] + " is not a parseable URL");

}

catch (IOException e) {

System.err.println(e);

}

}// endif } // end main

} // end SourceViewer3

Aside from the one line where the URLStreamHandlerFactory is set, this is almost exactly like the earlier SourceViewer program of Example 7.5 in Chapter 7. For instance, here the program reads from a finger URL:

Ngoài một dòng đặt URLStreamHandlerFactory, điều này gần giống như chương trình SourceViewer trước đó của Ví dụ 7.5 trong Chương 7. Ví dụ: ở đây chương trình đọc từ một URL ngón tay:

D:\JAVA\JNP2\examples\15>j**ava SourceViewer3 finger://rama.poly.edu/** Login Name TTY Idle When Where

nadats Nabeel Datsun

128.238.213.227

marcus Marcus Tullius

128.238.10.177

marcus Marcus Tullius

128.238.10.177

wri Weber Research Insti pts/10

rama.poly.edu

55 Fri 16:54

20 Thu 12:12

2:24 Thu 16:42

55 Fri 13:26

pts/0

\*pts/1

\*pts/5

jbjovi John B. Jovien pts/9 25d Mon 14:54

128.238.213.229

Here it reads from a daytime URL:

% **java SourceViewer3 daytime://tock.usno.navy.mil/** <html><head><title>The Time at tock.usno.navy.mil</title></head><body><h1>Fri Oc t 29 21:22:49 1999

</h1></body></html>

However, it still works with all the usual protocol handlers that come bundled with the JDK. For instance here are the first few lines of output when it reads from an http URL:

Tuy nhiên, nó vẫn hoạt động với tất cả các trình xử lý giao thức thông thường đi kèm với JDK. Ví dụ: đây là một vài dòng đầu ra khi nó đọc từ một URL http:

% **java SourceViewer3 http://www.oreilly.com/oreilly/about.html** <HTML>

<HEAD>

<TITLE>About O'Reilly &amp; Associates</TITLE>

</HEAD>

<BODY LINK="#770000" VLINK="#0000AA" BGCOLOR="#ffffff">

<table border=0 cellspacing=0 cellpadding=0 width=515>

<tr>

<td>

<img src="http://www.oreilly.com/graphics\_new/generic\_ora\_header\_wide.gif" width="515" height="37" ALT="O'Reilly and Associates">