CS1401 - INTERNET PROGRAMMING 3 0 0 100

AIM

To explain Internet Programming concepts and related programming and scripting languages.

OBJECTIVES

- To describe basic Internet Protocols.
- Explain JAVA and HTML tools for Internet programming.
- Describe scripting languages – Java Script.
- Explain dynamic HTML programming.
- Explain Server Side Programming tools.

UNIT I - BASIC NETWORK AND WEB CONCEPTS 9

Internet standards – TCP and UDP protocols – URLs – MIME – CGI – Introduction to SGML.

UNIT II - JAVA PROGRAMMING 9

Java basics – I/O streaming – files – Looking up Internet Address - Socket programming – client/server programs – E-mail client – SMTP - POP3 programs – web page retrieval – protocol handlers – content handlers - applets – image handling - Remote Method Invocation.

UNIT III - SCRIPTING LANGUAGES 9

HTML – forms – frames – tables – web page design - JavaScript introduction – control structures – functions – arrays – objects – simple web applications

UNIT IV - DYNAMIC HTML 9

Dynamic HTML – introduction – cascading style sheets – object model and collections – event model – filters and transition – data binding – data control – ActiveX control – handling of multimedia data

UNIT V - SERVER SIDE PROGRAMMING 9

Servlets – deployment of simple servlets – web server (Java web server / Tomcat / Web logic) – HTTP GET and POST requests – session tracking – cookies – JDBC – simple web applications – multi-tier applications.

TOTAL : 45

TEXT BOOKS

1. Deitel, Deitel and Nieto, “Internet and World Wide Web – How to program”, Pearson Education Publishers, 2000.
2. Elliotte Rusty Harold, “Java Network Programming”, O’Reilly Publishers, 2002

REFERENCES

1. R. Krishnamoorthy & S. Prabhu, “Internet and Java Programming”, New Age International Publishers, 2004.
2. Thomno A. Powell, “The Complete Reference HTML and XHTML”, fourth edition, Tata McGraw Hill, 2003.
3. Naughton, “The Complete Reference – Java2”, Tata McGraw-Hill, 3rd edition, 1999.

- x - x - x -

UNIT - I

BASIC NETWORK AND WEB CONCEPTS

CHAPTER - 1

BASIC NETWORK CONCEPTS

This chapter covers the background networking concepts you need to understand before writing networked programs in Java (or, for that matter, in any language). Moving from the most general to the most specific, it explains what you need to know about networks in general, IP and TCP/IP-based networks in particular, and the Internet. This chapter doesn't try to teach you how to wire a network or configure a router, but you will learn what you need to know to write applications that communicate across the Internet. Topics covered in this chapter include the definition of network, the TCP/IP layer model, the IP, TCP, and UDP protocols, firewalls and proxy servers, the Internet, and the Internet standardization process. Experienced network gurus may safely skip this chapter.

1.1 Networks

A network is a collection of computers and other devices that can send data to and receive data from each other, more or less in real time. A network is often connected by wires, and the bits of data are turned into electromagnetic waves that move through the wires. However, wireless networks transmit data through infrared light and microwaves, and many long-distance transmissions are now carried over fiber optic cables that send visible light through glass filaments. There's nothing sacred about any particular physical medium for the transmission of data. Theoretically, data could be transmitted by coal-powered computers that send smoke signals to each other. The response time (and environmental impact) of such a network would be rather poor.

Each machine on a network is called a node. Most nodes are computers, but printers, routers, bridges, gateways, dumb terminals, and Coca-Cola™ machines can also be nodes. You might use Java to interface with a Coke machine but otherwise, you'll mostly talk to other computers. Nodes that are fully functional computers are also called hosts. We will use the word node to refer to any device on the network, and the word host to refer to a node that is a general-purpose computer.

Every network node has an address, a series of bytes that uniquely identify it. You can think of this group of bytes as a number, but in general the number of bytes in an address or the ordering of those bytes (big endian or little endian) is not guaranteed to match any primitive numeric data type in Java. The more bytes there are in each address, the more addresses there are available and the more devices that can be connected to the network simultaneously.

Addresses are assigned differently on different kinds of networks. AppleTalk addresses are chosen randomly at startup by each host. The host then checks to see if any other machine on the network is using that address. If another machine is using the address, the host randomly chooses another, checks to see if that address is already in use, and so on until it gets one that isn't being used. Ethernet addresses are attached to the physical Ethernet hardware. Manufacturers of Ethernet hardware use pre-assigned manufacturer codes to make sure there are no conflicts between the addresses in their hardware and the addresses of other manufacturer's hardware. Each manufacturer is responsible for making sure it doesn't ship two Ethernet cards with the same address. Internet addresses are normally assigned to a computer by the organization that is responsible for it. However, the addresses that an organization is allowed to choose for its computers are assigned by the organization's Internet Service Provider (ISP). ISPs get their IP addresses from one of four regional Internet Registries (the registry for North America is ARIN, the American Registry for Internet Numbers, at http://www.arin.net/), which are in turn assigned IP addresses by the Internet Corporation for Assigned Names and Numbers (ICANN, at http://www.icann.org/).

On some kinds of networks, nodes also have names that help human beings identify them. At a set moment in time, a particular name normally refers to exactly one address. However, names are not locked to addresses. Names can change while addresses stay the same or addresses can change while the names stay the same. It is not uncommon for one address to have several names and it is possible, though somewhat less common, for one name to refer to several different addresses.

All modern computer networks are packet-switched networks: data traveling on the network is broken into chunks called packets and each packet is handled separately. Each packet contains information about who sent it and where it's going. The most important advantage of breaking data into individually addressed packets is that packets from many ongoing exchanges can travel on one wire, which makes it much cheaper to build a network: many computers can share the same wire without interfering. (In contrast, when you make a local telephone call within the same exchange, you have essentially reserved a wire from your phone to the phone of the person you're calling. When all the wires are in use, as sometimes happens during a major emergency or holiday, not everyone who picks up a phone will get a dial tone. If you stay on the line, you'll eventually get a dial tone when a line becomes free. In some countries with worse phone service than the United States, it's not uncommon to have to wait half an hour or more for a dial tone.) Another advantage of packets is that checksums can be used to detect whether a packet was damaged in transit.

We're still missing one important piece: some notion of what computers need to say to pass data back and forth. A protocol is a precise set of rules defining how computers communicate: the format of addresses, how data is split into packets, and so on. There are many different protocols defining different aspects of network communication. For example, the Hypertext Transfer Protocol (HTTP) defines how web browsers and servers communicate; at the other end of the spectrum, the IEEE 802.3 standard defines a protocol for how bits are encoded as electrical signals on a particular type of wire (among other protocols). Open, published protocol standards allow software and equipment from different vendors to communicate with each other: your web browser doesn't care whether any given server is a Unix workstation, a Windows box, or a Macintosh, because the server and the browser speak the same HTTP protocol regardless of platform.

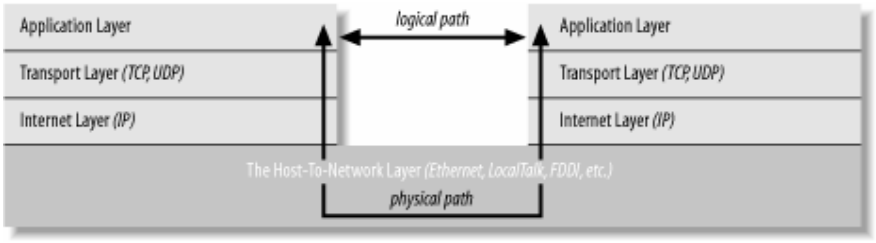
1.2 The Layers of a Network

Sending data across a network is a complex operation that must be carefully tuned to the physical characteristics of the network as well as the logical character of the data being sent. Software that sends data across a network must understand how to avoid collisions between packets, convert digital data to analog signals, detect and correct errors, route packets from one host to another, and more. The process becomes even more complicated when the requirement to support multiple operating systems and heterogeneous network cabling is added.

To make this complexity manageable and hide most of it from the application developer and end user, the different aspects of network communication are separated into multiple layers. Each layer represents a different level of abstraction between the physical hardware (e.g., the wires and electricity) and the information being transmitted. Each layer has a strictly limited function. For instance, one layer may be responsible for routing packets, while the layer above it is responsible for detecting and requesting retransmission of corrupted packets. In theory, each layer only talks to the layers immediately above and immediately below it. Separating the network into layers lets you modify or even replace the software in one layer without affecting the others, as long as the interfaces between the layers stay the same.

There are several different layer models, each organized to fit the needs of a particular kind of network. This book uses the standard TCP/IP four-layer model appropriate for the Internet, shown in Figure 1-1. In this model, applications like Internet Explorer and Eudora run in the application layer and talk only to the transport layer. The transport layer talks only to the application layer and the internet layer. The internet layer in turn talks only to the host-to-network layer and the transport layer, never directly to the application layer. The host-to-network layer moves the data across the wires, fiber optic cables, or other medium to the host-to-network layer on the remote system, which then moves the data up the layers to the application on the remote system.

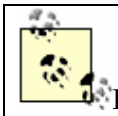
Figure 1-1. The layers of a network



For example, when a web browser sends a request to a web server to retrieve a page, the browser is actually only talking to the transport layer on the local client machine. The transport layer breaks the request up into TCP segments, adds some sequence numbers and checksums to the data, and then passes the request to the local internet layer. The internet layer fragments the segments into IP datagrams of the necessary size for the local network and passes them to the host-to-network layer for transmission onto the wire. The host-to-network layer encodes the digital data as analog signals appropriate for the particular physical medium and sends the request out the wire where it will be read by the host-to-network layer of the remote system to which it's addressed.

The host-to-network layer on the remote system decodes the analog signals into digital data then passes the resulting IP datagrams to the server's internet layer. The internet layer does some simple checks to see that the IP datagrams aren't corrupt, reassembles them if they've been fragmented, and passes them to the server's transport layer. The server's transport layer checks to see that all the data arrived and requests retransmission of any missing or corrupt pieces. (This request actually goes back down through the server's internet layer, through the server's host-to-network layer, and back to the client system, where it bubbles back up to the client's transport layer, which retransmits the missing data back down through the layers. This is all transparent to the application layer.) Once the server's transport layer has received enough contiguous, sequential datagrams, it reassembles them and writes them onto a stream read by the web server running in the server application layer. The server responds to the request and sends its response back down through the layers on the server system for transmission back across the Internet and delivery to the web client.

As you can guess, the real process is much more elaborate. The host-to-network layer is by far the most complex, and a lot has been deliberately hidden. For example, it's entirely possible that data sent across the Internet will pass through several routers and their layers before reaching its final destination. However, 90% of the time your Java code will work in the application layer and only need to talk to the transport layer. The other 10% of the time, you'll be in the transport layer and talking to the application layer or the internet layer. The complexity of the host-to-network layer is hidden from you; that's the point of the layer model.

 If you read the network literature, you're likely to encounter an alternative seven-layer model called the Open Systems Interconnection Reference Model (OSI). For network programs in Java, the OSI model is overkill. The biggest difference between the OSI model and the TCP/IP model used in this book is that the OSI model splits the host-to-network layer into data link and physical layers and inserts presentation and session layers in between the application and transport layers. The OSI model is more general and better suited for non-TCP/IP networks, although most of the time it's still overly complex. In any case, Java's network classes only work on TCP/IP networks and always in the application or transport layers, so for the purposes of this book, absolutely nothing is gained by using the more complicated OSI model.

To the application layer, it seems as if it is talking directly to the application layer on the other system; the network creates a logical path between the two application layers. It's easy to understand the logical path if you think about an IRC chat session. Most participants in an IRC chat would say that they're talking to another person. If you really push them, they might say that they're talking to their computer (really the application layer), which is talking to the other person's computer, which is talking to the other person. Everything more than one layer deep is effectively invisible, and that is exactly the way it should be. Let's consider each layer in more detail.

1.2.1 The Host-to-Network Layer

As a Java programmer, you're fairly high up in the network food chain. A lot happens below your radar. In the standard reference model for IP-based Internets (the only kind of network Java really understands), the hidden parts of the network belong to the host-to-network layer (also known as the link layer, data link layer, or network interface layer). The host-to-network layer defines how a particular network interface—such as an Ethernet card or a PPP connection—sends IP datagrams over its physical connection to the local network and the world.

The part of the host-to-network layer made up of the hardware that connects different computers (wires, fiber optic cables, microwave relays, or smoke signals) is sometimes called the physical layer of the network. As a Java programmer, you don't need to worry about this layer unless something goes wrong—the plug falls out of the back of your computer, or someone drops a backhoe through the T-1 line between you and the rest of the world. In other words, Java never sees the physical layer.

For computers to communicate with each other, it isn't sufficient to run wires between them and send electrical signals back and forth. The computers have to agree on certain standards for how those signals are interpreted. The first step is to determine how the packets of electricity or light or smoke map into bits and bytes of data. Since the physical layer is analog, and bits and bytes are digital, this process involves a digital-to-analog conversion on the sending end and an analog-to-digital conversion on the receiving end.

Since all real analog systems have noise, error correction and redundancy need to be built into the way data is translated into electricity. This is done in the data link layer. The most common data link layer is Ethernet. Other popular data link layers include TokenRing, PPP, and Wireless Ethernet (802.11). A specific data link layer requires specialized hardware. Ethernet cards won't communicate on a TokenRing network, for example. Special devices called gateways convert information from one type of data link layer, such as Ethernet, to another, such as TokenRing. As a Java programmer, the data link layer does not affect you directly. However, you can sometimes optimize the data you send in the application layer to match the native packet size of a particular data link layer, which can have some affect on performance. This is similar to matching disk reads and writes to the native block size of the disk. Whatever size you choose, the program will still run, but some sizes let the program run more efficiently than others, and which sizes these are can vary from one computer to the next.

1.2.2 The Internet Layer

The next layer of the network, and the first that you need to concern yourself with, is the internet layer. In the OSI model, the internet layer goes by the more generic name network layer. A network layer protocol defines how bits and bytes of data are organized into the larger groups called packets, and the addressing scheme by which different machines find each other. The Internet Protocol (IP) is the most widely used network layer protocol in the world and the only network layer protocol Java understands. IP is almost exclusively the focus of this book. Other, semi-common network layer protocols include Novell's IPX, and IBM and Microsoft's NetBEUI, although nowadays most installations have replaced these protocols with IP. Each network layer protocol is independent of the lower layers. IP, IPX, NetBEUI, and other protocols can each be used on Ethernet, Token Ring, and other data link layer protocol networks, each of which can themselves run across different kinds of physical layers.

Data is sent across the internet layer in packets called datagrams. Each IP datagram contains a header between 20 and 60 bytes long and a payload that contains up to 65,515 bytes of data. (In practice, most IP datagrams are much smaller, ranging from a few dozen bytes to a little more than eight kilobytes.) The header of each IP datagram contains these items, in this order:

4-bit version number

> Always 0100 (decimal 4) for current IP; will be changed to 0110 (decimal 6) for IPv6, but the entire header format will also change in IPv6.

4-bit header length

> An unsigned integer between 0 and 15 specifying the number of 4-byte words in the header; since the maximum value of the header length field is 1111 (decimal 15), an IP header can be at most 60 bytes long.

1-byte type of service

> A 3-bit precedence field that is no longer used, four type-of-service bits (minimize delay, maximize throughput, maximize reliability, minimize monetary cost) and a zero bit. Not all service types are compatible. Many computers and routers simply ignore these bits.

2-byte datagram length

> An unsigned integer specifying the length of the entire datagram, including both header and payload.

2-byte identification number

> A unique identifier for each datagram sent by a host; allows duplicate datagrams to be detected and thrown away.

3-bit flags

The first bit is 0; the second bit is 0 if this datagram may be fragmented, 1 if it may not be; and the third bit is 0 if this is the last fragment of the datagram, 1 if there are more fragments.

13-bit fragment offset

In the event that the original IP datagram is fragmented into multiple pieces, this field identifies the position of this fragment in the original datagram.

1-byte time-to-live (TTL)

Number of nodes through which the datagram can pass before being discarded; used to avoid infinite loops.

1-byte protocol

6 for TCP, 17 for UDP, or a different number between 0 and 255 for each of more than 100 different protocols (some quite obscure); see http://www.iana.org/assignments/protocol-numbers for the complete current list.

2-byte header checksum

A checksum of the header only (not the entire datagram) calculated using a 16-bit one's complement sum.

4-byte source address

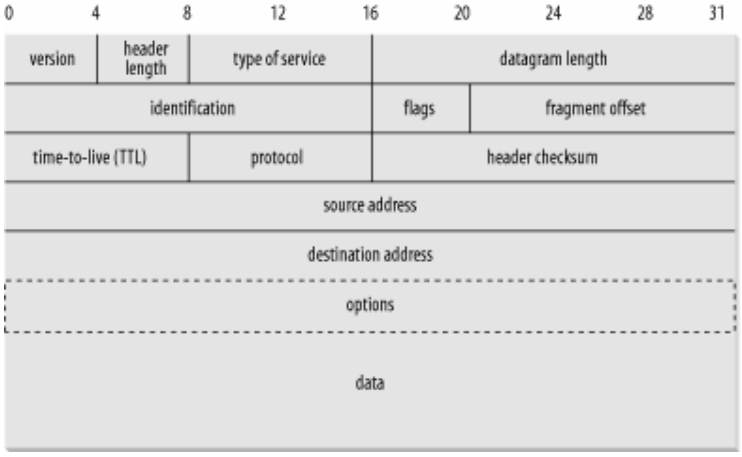
The IP address of the sending node.

4-byte destination address

The IP address of the destination node.

In addition, an IP datagram header may contain between 0 and 40 bytes of optional information, used for security options, routing records, timestamps, and other features Java does not support. Consequently, we will not discuss them here. The interested reader is referred to TCP/IP Illustrated, Volume 1: The Protocols, by W. Richard Stevens (Addison Wesley), for more details on these fields. Figure 1-2 shows how the different quantities are arranged in an IP datagram. All bits and bytes are big-endian; most significant to least significant runs left to right.

Figure 1-2. The structure of an IPv4 datagram



1.2.3 The Transport Layer

Raw datagrams have some drawbacks. Most notably, there's no guarantee that they will be delivered. Even if they are delivered, they may have been corrupted in transit. The header checksum can only detect corruption in the header, not in the data portion of a datagram. Finally, even if the datagrams arrive uncorrupted, they do not necessarily arrive in the order in which they were sent. Individual datagrams may follow different routes from source to destination. Just because datagram A is sent before datagram B does not mean that datagram A will arrive before datagram B.

The transport layer is responsible for ensuring that packets are received in the order they were sent and making sure that no data is lost or corrupted. If a packet is lost, the transport layer can ask the sender to retransmit the packet. IP networks implement this by adding an additional header to each datagram that contains more information. There are two primary protocols at this level. The first, the Transmission Control Protocol (TCP), is a high-overhead protocol that allows for retransmission of lost or corrupted data and delivery of bytes in the order they were sent. The second protocol, the User Datagram Protocol (UDP), allows the receiver to detect corrupted packets but does not guarantee that packets are delivered in the correct order (or at all). However, UDP is often much faster than TCP. TCP is called a reliable protocol; UDP is an unreliable protocol. Later, we'll see that unreliable protocols are much more useful than they sound.

1.2.4 The Application Layer

The layer that delivers data to the user is called the application layer. The three lower layers all work together to define how data is transferred from one computer to another. The application layer decides what to do with the data after it's transferred. For example, an application protocol like HTTP (for the World Wide Web) makes sure that your web browser knows to display a graphic image as a picture, not a long stream of numbers. The application layer is where most of the network parts of your programs spend their time. There is an entire alphabet soup of application layer protocols; in addition to HTTP for the Web, there are SMTP, POP, and IMAP for email; FTP, FSP, and TFTP for file transfer; NFS for file access; NNTP for news transfer; Gnutella, FastTrack, and Freenet for file sharing; and many, many more. In addition, your programs can define their own application layer protocols as necessary.

1.3 IP, TCP, and UDP

IP, the Internet protocol, has a number of advantages over competing protocols such as AppleTalk and IPX, most stemming from its history. It was developed with military sponsorship during the Cold War, and ended up with a lot of features that the military was interested in. First, it had to be robust. The entire network couldn't stop functioning if the Soviets nuked a router in Cleveland; all messages still had to get through to their intended destinations (except those going to Cleveland, of course). Therefore IP was designed to allow multiple routes between any two points and to route packets of data around damaged routers.

Second, the military had many different kinds of computers, and all of them had to be able to talk to each other. Therefore the IP had to be open and platform-independent; it wasn't good enough to have one protocol for IBM mainframes and another for PDP-11s. The IBM mainframes needed to talk to the PDP-11s and any other strange computers that might be lying around.

Since there are multiple routes between two points, and since the quickest path between two points may change over time as a function of network traffic and other factors (such as the existence of Cleveland), the packets that make up a particular data stream may not all take the same route. Furthermore, they may not arrive in the order they were sent, if they even arrive at all. To improve on the basic scheme, TCP was layered on top of IP to give each end of a connection the ability to acknowledge receipt of IP packets and request retransmission of lost or corrupted packets. Furthermore, TCP allows the packets to be put back together on the receiving end in the same order they were sent.

TCP, however, carries a fair amount of overhead. Therefore, if the order of the data isn't particularly important and if the loss of individual packets won't completely corrupt the data stream, packets are sometimes sent without the guarantees that TCP provides. This is accomplished through the use of the UDP protocol. UDP is an unreliable protocol that does not guarantee that packets will arrive at their destination or that they will arrive in the same order they were sent. Although this would be a problem for uses such as file transfer, it is perfectly acceptable for applications where the loss of some data would go unnoticed by the end user. For example, losing a few bits from a video or audio signal won't cause much degradation; it would be a bigger problem if you had to wait for a protocol like TCP to request a retransmission of missing data. Furthermore, error-correcting codes can be built into UDP data streams at the application level to account for missing data.

A number of other protocols can run on top of IP. The most commonly requested is ICMP, the Internet Control Message Protocol, which uses raw IP datagrams to relay error messages between hosts. The best-known use of this protocol is in the ping program. Java does not support ICMP nor does it allow the sending of raw IP datagrams (as opposed to TCP segments or UDP datagrams). The only protocols Java supports are TCP and UDP, and application layer protocols built on top of these. All other transport layer, internet layer, and lower layer protocols such as ICMP, IGMP, ARP, RARP, RSVP, and others can only be implemented in Java programs by using native code.

1.3.1 IP Addresses and Domain Names

As a Java programmer, you don't need to worry about the inner workings of IP, but you do need to know about addressing. Every computer on an IPv4 network is identified by a four-byte number. This is normally written in a dotted quad format like 199.1.32.90, where each of the four numbers is one unsigned byte ranging in value from 0 to 255. Every computer attached to an IPv4 network has a unique four-byte address. When data is transmitted across the network, the packet's header includes the address of the machine for which the packet is intended (the destination address) and the address of the machine that sent the packet (the source address). Routers along the way choose the best route to send the packet along by inspecting the destination address. The source address is included so the recipient will know who to reply to.

There are a little more than four billion possible IP addresses, not even one for every person on the planet, much less for every computer. To make matters worse, the addresses aren't allocated very efficiently. A slow transition is under way to IPv6, which will use 16-byte addresses. This provides enough IP addresses to identify every person, every computer, and indeed every atom on the planet. IPv6 addresses are customarily written in eight blocks of four hexadecimal digits separated by colons, such as FEDC:BA98:7654:3210:FEDC:BA98:7654:3210. Leading zeros do not need to be written. A double colon, at most one of which may appear in any address, indicates multiple zero blocks. For example, FEDC:0000:0000:0000:00DC:0000:7076:0010 could be written more compactly as FEDC::DC:0:7076:10. In mixed networks of IPv6 and IPv4, the last four bytes of the IPv6 address are sometimes written as an IPv4 dotted quad address. For example, FEDC:BA98:7654:3210:FEDC:BA98:7654:3210 could be written as FEDC:BA98:7654:3210:FEDC:BA98:118.84.50.16. IPv6 is only supported in Java 1.4 and later. Java 1.3 and earlier only support four-byte addresses.

Although computers are very comfortable with numbers, human beings aren't very good at remembering them. Therefore the Domain Name System (DNS) was developed to translate hostnames that humans can remember (like www.oreilly.com) into numeric Internet addresses (like 208.201.239.37). When Java programs access the network, they need to process both these numeric addresses and their corresponding hostnames. Methods for doing this are provided by the `java.net.InetAddress` class, which is discussed in Chapter 6.

Some computers, especially servers, have fixed addresses. Others, especially clients on local area networks and dial-up connections, receive a different address every time they boot up, often provided by a DHCP server or a PPP server. This is not especially relevant to your Java programs. Mostly you just need to remember that IP addresses may change over time, and not write any code that relies on a system having the same IP address. For instance, don't serialize the local IP address when saving application state. Instead, look it up fresh each time your program starts. It's also possible, although less likely, for an IP address to change while the program is running (for instance, if a dialup connection hangs up and then reconnects), so you may want to check the current IP address every time you need it rather than caching it. Otherwise, the difference between a dynamically and manually assigned address is not significant to Java programs.

1.3.2 Ports

Addresses would be all you needed if each computer did no more than one thing at a time. However, modern computers do many different things at once. Email needs to be separated from FTP requests, which need to be separated from web traffic. This is accomplished through *ports*. Each computer with an IP address has several thousand logical ports (65,535 per transport layer protocol, to be precise). These are purely abstractions in the computer's memory and do not represent anything physical, like a serial or parallel port. Each port is identified by a number between 1 and 65,535. Each port can be allocated to a particular service.

For example, HTTP, the underlying protocol of the Web, generally uses port 80. We say that a web server listens on port 80 for incoming connections. When data is sent to a web server on a particular machine at a particular IP address, it is also sent to a particular port (usually port 80) on that machine. The receiver checks each packet it sees for the port and sends the data to any programs that are listening to the specified port. This is how different types of traffic are sorted out.

Port numbers between 1 and 1,023 are reserved for well-known services like finger, FTP, HTTP, and IMAP. On Unix systems, including Linux and Mac OS X, only programs running as root can receive data from these ports, but all programs may send data to them. On Windows and Mac OS 9, any program may use these ports without special privileges. Table 1-1 shows the well-known ports for the protocols that are discussed in this book. These assignments are not absolutely guaranteed; in particular, web servers often run on ports other than 80, either because multiple servers need to run on the same machine or because the person who installed the server doesn't have the root privileges needed to run it on port 80. On Unix systems, a fairly complete listing of assigned ports is stored in the file */etc/services*.

Table 1-1. Well-known port assignments

Protocol	Port	Protocol	Purpose
echo	7	TCP/UDP	Echo is a test protocol used to verify that two machines are able to connect by having one echo back the other's input.
discard	9	TCP/UDP	Discard is a less useful test protocol in which all data received by the server is ignored.
daytime	13	TCP/UDP	Provides an ASCII representation of the current time on the server.
FTP data	20	TCP	FTP uses two well-known ports. This port is used to transfer files.
FTP	21	TCP	This port is used to send FTP commands like `put` and `get`.
SSH	22	TCP	Used for encrypted, remote logins.
telnet	23	TCP	Used for interactive, remote command-line sessions.
smtp	25	TCP	The Simple Mail Transfer Protocol is used to send email between machines.
time	37	TCP/UDP	A time server returns the number of seconds that have elapsed on the server since midnight, January 1, 1900, as a four-byte, signed, big-endian integer.
whois	43	TCP	A simple directory service for Internet network administrators.
finger	79	TCP	A service that returns information about a user or users on the local system.
HTTP	80	TCP	The underlying protocol of the World Wide Web.
POP3	110	TCP	Post Office Protocol Version 3 is a protocol for the transfer of accumulated email from the host to sporadically connected clients.
NNTP	119	TCP	Usenet news transfer; more formally known as the "Network News Transfer Protocol".
IMAP	143	TCP	Internet Message Access Protocol is a protocol for accessing mailboxes stored on a server.
RMI Registry	1099	TCP	The registry service for Java remote objects. This will be discussed in Chapter 18.

1.4 The Internet

The Internet is the world's largest IP-based network. It is an amorphous group of computers in many different countries on all seven continents (Antarctica included) that talk to each other using the IP protocol. Each computer on the Internet has at least one unique IP address by which it can be identified. Most of them also have at least one name that maps to that IP address. The Internet is not owned by anyone, although pieces of it are. It is not governed by anyone, which is not to say that some governments don't try. It is simply a very large collection of computers that have agreed to talk to each other in a standard way.

The Internet is not the only IP-based network, but it is the largest one. Other IP networks are called internets with a little i: for example, a corporate IP network that is not connected to the Internet. Intranet is a current buzzword that loosely describes corporate practices of putting lots of data on internal web servers.

Unless you're working in a high security environment that's physically disconnected from the broader network, it's likely that the internet you'll be using is the Internet. To make sure that hosts on different networks on the Internet can communicate with each other, a few rules need to be followed that don't apply to purely internal internets. The most important rules deal with the assignment of addresses to different organizations, companies, and individuals. If everyone picked the Internet addresses they wanted at random, conflicts would arise almost immediately when different computers showed up on the Internet with the same address.

1.4.1 Internet Address Classes

To avoid this problem, blocks of IPv4 addresses are assigned to Internet Service Providers (ISPs) by their regional Internet registry. When a company or an organization wants to set up an IP-based network connected to the Internet, their ISP gives them a block of addresses. Traditionally, these blocks come in three sizes called Class A, Class B, and Class C. A Class C address block specifies the first three bytes of the address; for example, 199.1.32. This allows room for 254 individual addresses from 199.1.32.1 to 199.1.32.254.[1] A class B address block only specifies the first two bytes of the addresses an organization may use; for instance, 167.1. Thus, a class B address has room for 65,024 different hosts (256 Class C size blocks times 254 hosts per Class C block). A class A address block only specifies the first byte of the address range—for instance, 18—and therefore has room for over 16 million nodes.

[1] Addresses with the last byte either .0 or .255 are reserved and should never actually be assigned to hosts.

 There are also Class D and E addresses. Class D addresses are used for IP multicast groups, and will be discussed at length in Chapter 14. Class D addresses all begin with the four bits 1110. Class E addresses begin with the five bits 11110 and are reserved for future extensions to the Internet.

There's no block with a size between a class A and a Class B, or Class B and a Class C. This has become a problem because there are many organizations with more than 254 computers connected to the Internet but less than 65,024. If each of these organizations gets a full Class B block, many addresses are wasted. There's a limited number of IPv4 addresses—about 4.2 billion, to be precise. That sounds like a lot, but it gets crowded quickly when you can easily waste fifty or sixty thousand addresses at a shot.

There are also many networks, such as the author's own personal basement-area network, that have a few to a few dozen computers but not 255. To more efficiently allocate the limited address space, Classless Inter-Domain Routing (CIDR) was invented. CIDR mostly (though not completely) replaces the whole A, B, C, D, E addressing scheme with one based on a specified numbers of prefix bits. These prefixes are generally written as /nn, where nn is a two-digit number specifying the number of bits in the network portion of the address. The number after the / indicates the number of fixed prefix bits. Thus, a /24 fixes the first 24 bits in the address, leaving 8 bits available to distinguish individual nodes. This allows 256 nodes, and is equivalent to an old style Class C. A /19 fixes 19 bits, leaving 13 for individual nodes within the network. It's equivalent to 32 separate Class C networks or an eighth of a Class B. A /28, generally the smallest you're likely to encounter in practice, leaves only four bits for identifying local nodes. It can handle networks with up to 16 nodes. CIDR also carefully specifies which address blocks are associated with which ISPs. This scheme helps keep Internet routing tables smaller and more manageable than they would be under the old system.

Several address blocks and patterns are special. All IPv4 addresses that begin with 10., 172.16. through 172.31., and 192.168. are deliberately unassigned. They can be used on internal networks, but no host using addresses in these blocks is allowed onto the global Internet. These non-routable addresses are useful for building private networks that can't be seen from the rest of the Internet or for building a large network when you've only been assigned a class C address block. IPv4 addresses beginning with 127 (most commonly 127.0.0.1) always mean the local loopback address. That is, these addresses always point to the local computer, no matter which computer you're running on. The hostname for this address is generally localhost. In IPv6 0:0:0:0:0:0:0:1 (a.k.a. ::1) is the loopback address. The address 0.0.0.0 always refers to the originating host, but may only be used as a source address, not a destination. Similarly, any IPv4 address that begins with 0.0 is assumed to refer to a host on the same local network.

1.4.2 Network Address Translation

For reasons of both security and address space conservation, many smaller networks, such as the author's home network, use network address translation (NAT). Rather than allotting even a /28, my ISP gives me a single address, 216.254.85.72. Obviously, that won't work for the dozen or so different computers and other devices running in my apartment at any one time. Instead, I assign each one of them a different address in the non-routable block 192.168.254.xxx. When they connect to the internet, they have to pass through a router my ISP sold me that translates the internal addresses into the external addresses.

The router watches my outgoing and incoming connections and adjusts the addresses in the IP packets. For an outgoing packet, it changes the source address to the router's external address (216.254.85.72 on my network). For an incoming packet, it changes the destination address to one of the local addresses, such as 192.168.254.12. Exactly how it keeps track of which connections come from and are aimed at which internal computers is not particularly important to a Java programmer. As long as your machines are configured properly, this process is mostly transparent to Java programs. You just need to remember that the external and internal addresses may not be the same. From outside my network, nobody can talk to my system at 192.168.254.12 unless I initiate the connection, or unless I configure my router to forward requests addressed to 216.254.85.72 to 192.168.254.12. If the router is safe, then the rest of the network is too. On the other hand, if someone does crack the router or one of the servers behind the router that is mapped to 216.254.85.72, I'm hosed. This is why I installed a firewall as the next line of defense.

1.4.3 Firewalls

There are some naughty people on the Internet. To keep them out, it's often helpful to set up one point of access to a local network and check all traffic into or out of that access point. The hardware and software that sit between the Internet and the local network, checking all the data that comes in or out to make sure it's kosher, is called a firewall. The firewall is often part of the router that connects the local network to the broader Internet and may perform other tasks, such as network address translation. Then again, the firewall may be a separate machine. Modern operating systems like Mac OS X and Red Hat Linux often have built-in personal firewalls that monitor just the traffic sent to that one machine. Either way, the firewall is responsible for inspecting each packet that passes into or out of its network interface and accepting it or rejecting it according to a set of rules.

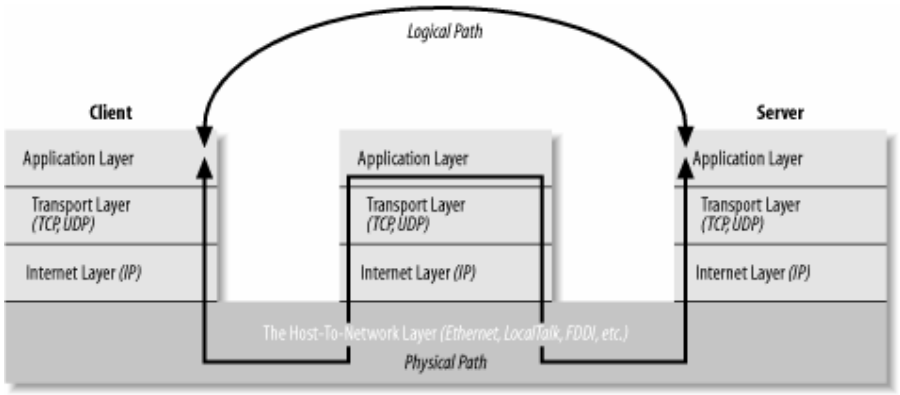
Filtering is usually based on network addresses and ports. For example, all traffic coming from the Class C network 193.28.25 may be rejected because you had bad experiences with hackers from that network in the past. Outgoing Telnet connections may be allowed, but incoming Telnet connections may not. Incoming connections on port 80 (web) may be allowed, but only to the corporate web server. More intelligent firewalls look at the contents of the packets to determine whether to accept or reject them. The exact configuration of a firewall—which packets of data are and are not allowed to pass through—depends on the security needs of an individual site. Java doesn't have much to do with firewalls—except in so far as they often get in your way.

1.4.4 Proxy Servers

Proxy servers are related to firewalls. If a firewall prevents hosts on a network from making direct connections to the outside world, a proxy server can act as a go-between. Thus, a machine that is prevented from connecting to the external network by a firewall would make a request for a web page from the local proxy server instead of requesting the web page directly from the remote web server. The proxy server would then request the page from the web server and forward the response back to the original requester. Proxies can also be used for FTP services and other connections. One of the security advantages of using a proxy server is that external hosts only find out about the proxy server. They do not learn the names and IP addresses of the internal machines, making it more difficult to hack into internal systems.

While firewalls generally operate at the level of the transport or internet layer, proxy servers normally operate at the application layer. A proxy server has a detailed understanding of some application level protocols, such as HTTP and FTP. (The notable exception are SOCKS proxy servers that operate at the transport layer, and can proxy for all TCP and UDP connections regardless of application layer protocol.) Packets that pass through the proxy server can be examined to ensure that they contain data appropriate for their type. For instance, FTP packets that seem to contain Telnet data can be rejected. Figure 1-3 shows how proxy servers fit into the layer model.

Figure 1-3. Layered connections through a proxy server



As long as all access to the Internet is forwarded through the proxy server, access can be tightly controlled. For instance, a company might choose to block access to www.playboy.com but allow access to www.microsoft.com. Some companies allow incoming FTP but disallow outgoing FTP so confidential data cannot be as easily smuggled out of the company. Other companies have begun using proxy servers to track their employees' web usage so they can see who's using the Internet to get tech support and who's using it to check out the Playmate of the Month. Such monitoring of employee behavior is controversial and not exactly an indicator of enlightened management techniques.

Proxy servers can also be used to implement local caching. When a file is requested from a web server, the proxy server first checks to see if the file is in its cache. If the file is in the cache, the proxy serves the file from the cache rather than from the Internet. If the file is not in the cache, the proxy server retrieves the file, forwards it to the requester, and stores it in the cache for the next time it is requested. This scheme can significantly reduce load on an Internet connection and greatly improve response time. America Online runs one of the largest farm of proxy servers in the world to speed the transfer of data to its users. If you look at a web server logfile, you'll probably find some hits from clients in the aol.com domain, but not as many as you'd expect given the more than twenty million AOL subscribers. That's because AOL proxy servers supply many pages out of their cache rather than re-requesting them for each user. Many other large ISPs do similarly.

The biggest problem with proxy servers is their inability to cope with all but a few protocols. Generally established protocols like HTTP, FTP, and SMTP are allowed to pass through, while newer protocols like Gnutella are not. (Some network administrators would consider this a feature.) In the rapidly changing world of the Internet, this is a significant disadvantage. It's a particular disadvantage for Java programmers because it limits the effectiveness of custom protocols. In Java, it's easy and often useful to create a new protocol that is optimized for your application. However, no proxy server will ever understand these one-of-a-kind protocols. Consequently, some developers have taken to tunneling their protocols through HTTP, most notably with SOAP. However, this has a significant negative impact on security. The firewall is normally there for a reason, not just to annoy Java programmers.

Applets that run in web browsers use the proxy server settings of the web browser itself, generally set in a dialog box (possibly hidden several levels deep in the preferences) like the one in Figure 1-4. Standalone Java applications can indicate the proxy server to use by setting the `socksProxyHost` and `socksProxyPort` properties (if you're using a SOCKS proxy server), or `http.proxySet`, `http.proxyHost`, `http.proxyPort`, `https.proxySet`, `https.proxyHost`, `https.proxyPort`, `ftpProxySet`, `ftpProxyHost`, `ftpProxyPort`, `gopherProxySet`, `gopherProxyHost`, and `gopherProxyPort` system properties (if you're using protocol-specific proxies). You can set system properties from the command line using the `-D` flag, like this:

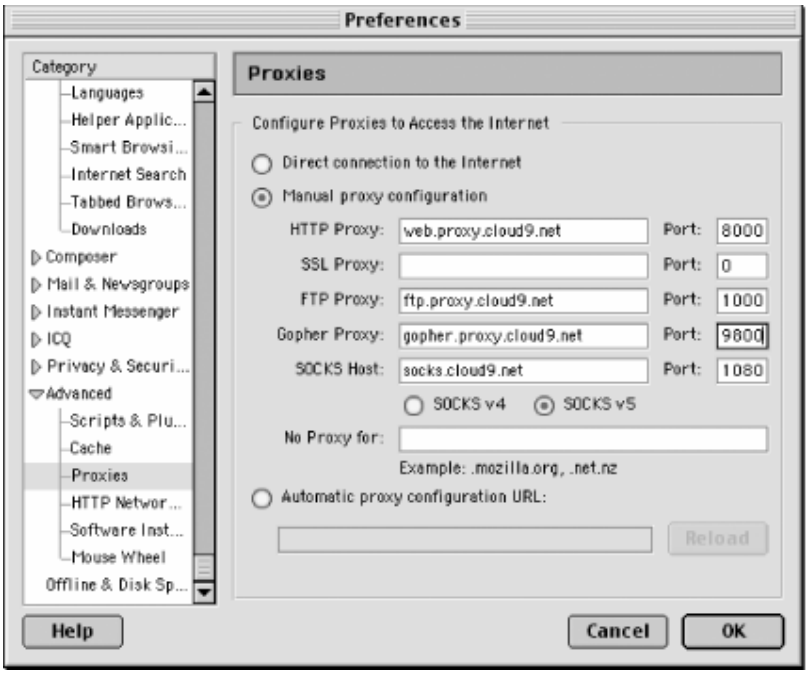
```
java -DsocksProxyHost=

socks.cloud9.net

 -DsocksProxyPort=

1080

  MyClass
```

You can use any other convenient means to set these system properties, such as including them in the *appletviewer.properties* file, like this:

```
ftpProxySet=true

ftpProxyHost=ftp.proxy.cloud9.net

ftpProxyPort=1000

gopherProxySet=true

gopherProxyHost=gopher.proxy.cloud9.net

gopherProxyPort=9800

http.proxySet=true
```

```
http.proxyHost=web.proxy.cloud9.net

http.proxyPort=8000

https.proxySet=true

https.proxyHost=web.proxy.cloud9.net

https.proxyPort=8001
```

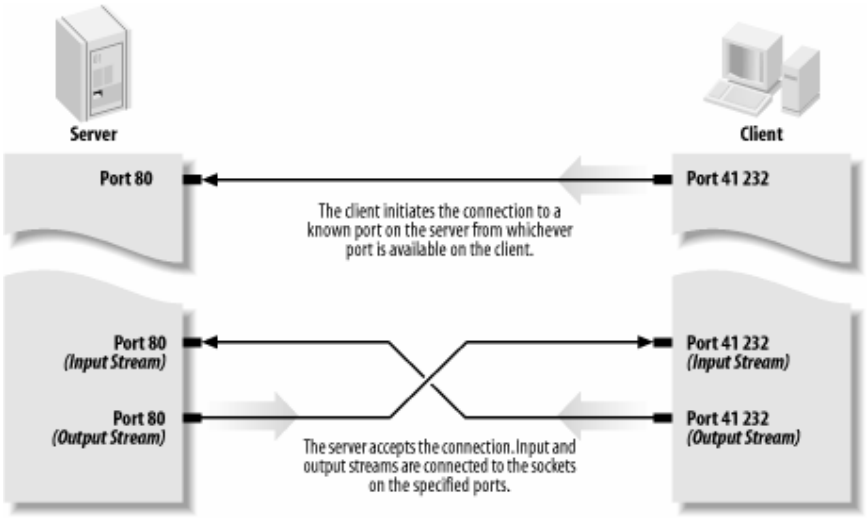
Figure 1-4. Netscape Navigator proxy server settings



1.5 The Client/Server Model

Most modern network programming is based on a client/server model. A client/server application typically stores large quantities of data on an expensive, high-powered server while most of the program logic and the user-interface is handled by client software running on relatively cheap personal computers. In most cases, a server primarily sends data while a client primarily receives it, but it is rare for one program to send or receive exclusively. A more reliable distinction is that a client initiates a conversation while a server waits for clients to start conversations with it. Figure 1-5 illustrates both possibilities. In some cases, the same program may be both a client and a server.

Figure 1-5. A client/server connection



Some servers process and analyze the data before sending the results to the client. Such servers are often referred to as "application servers" to distinguish them from the more common file servers and database servers. A file or database server will retrieve information and send it to a client, but it won't process that information. In contrast, an application server might look at an order entry database and give the clients reports about monthly sales trends. An application server is not a server that serves files that happen to be applications.

You are already familiar with many examples of client/server systems. In 2004, the most popular client/server system on the Internet is the Web. Web servers like Apache respond to requests from web clients like Firefox. Data is stored on the web server and is sent out to the clients that request it. Aside from the initial request for a page, almost all data is transferred from the server to the client, not from the client to the server. Web servers that use CGI programs double as application and file servers. FTP is an older service that fits the client/server model. FTP uses different application protocols and different software, but is still split into FTP servers that send files and FTP clients that receive files. People often use FTP to upload files from the client to the server, so it's harder to say that the data transfer is primarily in one direction, but it is still true that an FTP client initiates the connection and the FTP server responds.

Not all applications fit easily into a client/server model. For instance, in networked games, it seems likely that both players will send data back and forth roughly equally (at least in a fair game). These sorts of connections are called peer-to-peer. The telephone system is the classic example of a peer-to-peer network. Each phone can either call another phone or be called by another phone. You don't have to buy one phone to send calls and another to receive them.

Java does not have explicit peer-to-peer communication in its core networking API (though Sun has implemented it in a separate open source project called JXTA). However, applications can easily offer peer-to-peer communications in several ways, most commonly by acting as both a server and a client. Alternately, the peers can communicate with each other through an intermediate server program that forwards data from one peer to the other peers. This is especially useful for applets with a security manager that restricts them from talking directly to each other.

1.6 Internet Standards

This book discusses several application-layer Internet protocols, most notably HTTP. However, this is not a book about those protocols and it tries not to say more than the minimum you need to know. If you need detailed information about any protocol, the definitive source is the standards document for the protocol.

While there are many standards organizations in the world, the two that produce most of the standards relevant to network programming and protocols are the Internet Engineering Task Force (IETF) and the World Wide Web Consortium (W3C). The IETF is a relatively informal, democratic body open to participation by any interested party. Its standards are based on "rough consensus and running code" and tend to follow rather than lead implementations. IETF standards include TCP/IP, MIME, and SMTP. The W3C, by contrast, is a vendor organization, controlled by dues-paying member corporations, that explicitly excludes participation by individuals. For the most part, the W3C tries to define standards in advance of implementation. W3C standards include HTTP, HTML, and XML.

1.6.1 IETF RFCs

IETF standards and near-standards are published as Internet drafts and requests for comments (RFCs). RFCs and Internet drafts range from informational documents of general interest to detailed specifications of standard Internet protocols like FTP. RFCs that document a standard or a proposed standard are published only with the approval of the Internet Engineering Steering Group (IESG) of the IETF. All IETF approved standards are RFCs, but not all RFCs are IETF standards. RFCs are available from many locations on the Internet, including http://www.faqs.org/rfc/ and http://www.ietf.org/rfc.html. For the most part RFCs, particularly standards-oriented RFCs, are very technical, turgid, and nearly incomprehensible. Nonetheless, they are often the only complete and reliable source of information about a particular protocol.

Most proposals for a standard begin when a person or group gets an idea and builds a prototype. The prototype is incredibly important. Before something can become an IETF standard, it must actually exist and work. This requirement ensures that IETF standards are at least feasible, unlike the standards promulgated by some other organizations. If the prototype becomes popular outside its original developers and if other organizations begin implementing their own versions of the protocol, a working group may be formed under the auspices of the IETF. This working group attempts to document the protocol in an Internet-Draft. Internet-Drafts are working documents and change frequently to reflect experience with the protocol. The experimental implementations and the Internet-Draft evolve in rough synchronization, until eventually the working group agrees that the protocol is ready to become a formal standard. At this point, the proposed specification is submitted to the IESG.

The proposal goes through six states or maturity levels as it follows the standardization track:

- Experimental
- Proposed standard
- Draft standard
- Standard
- Informational
- Historic

For some time after the proposal is submitted, it is considered experimental. The experimental stage does not imply that the protocol is not solid or that it is not widely used; unfortunately, the standards process usually lags behind de facto acceptance of the standard. If the IESG likes the experimental standard or it is in widespread use, the IESG will assign it an RFC number and publish it as an experimental RFC, generally after various changes.

If the experimental standard holds up well in further real world testing, the IESG may advance it to the status of proposed standard. A proposed standard is fairly loose, and is based on the experimental work of possibly as little as one organization. Changes may still be made to a protocol in this stage.

Once the bugs appear to have been worked out of a proposed standard and there are at least two independent implementations, the IESG may recommend that a proposed standard be promoted to a draft standard. A draft standard will probably not change too much before eventual standardization unless major flaws are found. The primary purpose of a draft standard is to clean up the RFC that documents the protocol and make sure the documentation conforms to actual practice, rather than to change the standard itself.

When a protocol completes this, it becomes an official Internet standard. It is assigned an STD number and is published as an STD in addition to an RFC. The absolute minimum time for a standard to be approved as such is 10 months, but in practice, the process almost always takes much longer. The commercial success of the Internet hasn't helped, since standards must now be worked out in the presence of marketers, vulture capitalists, lawyers, NSA spooks, and others with vested interests in seeing particular technologies succeed or fail. Therefore, many of the "standards" that this book references are in either the experimental, proposed, or draft stage. As of publication, there are over 3,800 RFCs. Less than one hundred of these have become STDs, and some of those that have are now obsolete. RFCs relevant to this book are detailed in Table 2-2.

Some RFCs that do not become standards are considered informational,. These include RFCs that specify protocols that are widely used but weren't developed within the normal Internet standards track, and haven't been through the formal standardization process. For example, NFS, originally developed by Sun, is described in the informational RFC 1813. Other informational RFCs provide useful information (like users' guides), but don't document a protocol. For example, RFC 1635, How to Use Anonymous FTP, is an informational RFC.

Finally, changing technology and increasing experience renders some protocols and their associated RFCs obsolete. These are classified as historic. Historic protocols include IMAP3 (replaced by IMAP4), POP2 (replaced by POP3), and Remote Procedure Call Version 1 (replaced by Remote Procedure Call Version 2).

In addition to its maturity level, a protocol has a requirement level. The possible requirement levels are:

Not recommended

Should not be implemented by anyone.

Limited use

May have to be implemented in certain unusual situations but won't be needed by most hosts. Mainly these are experimental protocols.

Elective

Can be implemented by anyone who wants to use the protocol. For example, RFC 2045, Multipurpose Internet Mail Extensions, is a Draft Elective Standard.

Recommended

Should be implemented by Internet hosts that don't have a specific reason not to implement it. Most protocols that you are familiar with (like TCP and UDP, SMTP for email, Telnet for remote login, etc.) are recommended.

Required

Must be implemented by all Internet hosts. There are very few required protocols. IP itself is one (RFC 791), but even protocols as important as TCP or UDP are only recommended. A standard is only required if it is absolutely essential to the functioning of a host on the Internet.

Table 1-2 lists the RFCs and STDs that provide formal documentation for the protocols discussed in this book.

Table 1-2. Selected Internet RFCs

RFC	Title	Maturity level	Requirement level	Description
RFC 3300 STD 1	Internet Official Protocol Standards	Standard	Required	Describes the standardization process and the current status of the different Internet protocols.
RFC 1122 RFC 1123	Host Requirements	Standard	Required	Documents the protocols that must be supported by all Internet hosts at different layers (data link layer, IP layer, transport layer, and application layer).

RFC	Title	Maturity level	Requirement level	Description
STD 3				
RFC 791 RFC 919 RFC 922 RFC 950 STD 5	Internet Protocol	Standard	Required	The IP internet layer protocol.
RFC 768 STD 6	User Datagram Protocol	Standard	Recommended	An unreliable, connectionless transport layer protocol.
RFC 792 STD 5	Internet Control Message Protocol (ICMP)	Standard	Required	An internet layer protocol that uses raw IP datagrams but is not supported by Java. Its most familiar use is the *ping* program.
RFC 793 STD 7	Transmission Control Protocol	Standard	Recommended	A reliable, connection-oriented, streaming transport layer protocol.
RFC 2821	Simple Mail Transfer Protocol	Proposed standard	Recommended	The application layer protocol by which one host transfers email to another host. This standard doesn't say anything about email user interfaces; it covers the mechanism for passing email from one computer to another.
RFC 822 STD 11	Format of Electronic Mail Messages	Standard	Recommended	The basic syntax for ASCII text email messages. MIME is designed to extend this to support binary data while ensuring that the messages transferred still conform to this standard.
RFC	Telnet	Standard	Recommended	An application-layer remote login service

RFC	Title	Maturity level	Requirement level	Description
854 RFC 855 STD 8	Protocol			for command-line environments based around an abstract network virtual terminal (NVT) and TCP.
RFC 862 STD 20	Echo Protocol	Standard	Recommended	An application-layer protocol that echoes back all data it receives over both TCP and UDP; useful as a debugging tool.
RFC 863 STD 21	Discard Protocol	Standard	Elective	An application layer protocol that receives packets of data over both TCP and UDP and sends no response to the client; useful as a debugging tool.
RFC 864 STD 22	Character Generator Protocol	Standard	Elective	An application layer protocol that sends an indefinite sequence of ASCII characters to any client that connects over either TCP or UDP; also useful as a debugging tool.
RFC 865 STD 23	Quote of the Day	Standard	Elective	An application layer protocol that returns a quotation to any user who connects over either TCP or UDP and then closes the connection.
RFC 867 STD 25	Daytime Protocol	Standard	Elective	An application layer protocol that sends a human-readable ASCII string indicating the current date and time at the server to any client that connects over TCP or UDP. This contrasts with the various NTP and Time Server protocols, which do not return data that can be easily read by humans.
RFC 868 STD 26	Time Protocol	Standard	Elective	An application layer protocol that sends the time in seconds since midnight, January 1, 1900 to a client connecting over TCP or UDP. The time is sent as a machine-readable, 32-bit signed integer. The standard is incomplete in that it does not specify how the integer is encoded in 32 bits, but in practice a two's complement, big-endian integer is used.

RFC	Title	Maturity level	Requirement level	Description
RFC 959 STD 9	File Transfer Protocol	Standard	Recommended	An optionally authenticated, two-socket application layer protocol for file transfer that uses TCP.
RFC 977	Network News Transfer Protocol	Proposed standard	Elective	The application layer protocol by which Usenet news is transferred from machine to machine over TCP; used by both news clients talking to news servers and news servers talking to each other.
RFC 1034 RFC 1035 STD 13	Domain Name System	Standard	Recommended	The collection of distributed software by which hostnames that human beings can remember, like www.oreilly.com, are translated into numbers that computers can understand, like 198.112.208.11. This STD defines how domain name servers on different hosts communicate with each other using UDP.
RFC 1112	Host Extensions for IP Multicasting	Standard	Recommended	The internet layer methods by which conforming systems can direct a single packet of data to multiple hosts. This is called multicasting; Java's support for multicasting is discussed in Chapter 14.
RFC 1153	Digest Message Format for Mail	Experimental	Limited use	A format for combining multiple postings to a mailing list into a single message.
RFC 1288	Finger Protocol	Draft standard	Elective	An application layer protocol for requesting information about a user at a remote site. It can be a security risk.
RFC 1305	Network Time Protocol (Version 3)	Draft standard	Elective	A more precise application layer protocol for synchronizing clocks between systems that attempts to account for network latency.
RFC 1738	Uniform Resource Locators	Proposed standard	Elective	Full URLs like http://www.amnesty.org/ and ftp://ftp.ibiblio.org/pub/multimedia/chinese-music/Dream_Of_Red_Mansion/HLM04.Handkerchief.au.
RFC 1808	Relative Uniform Resource Locators	Proposed standard	Elective	Partial URLs like /javafaq/books/ and ../examples/07/index.html used as values of the HREF attribute of an HTML A element.

RFC	Title	Maturity level	Requirement level	Description
RFC 1939 STD 53	Post Office Protocol, Version 3	Standard	Elective	An application-layer protocol used by sporadically connected email clients such as Eudora to retrieve mail from a server over TCP.
RFC 1945	Hypertext Transfer Protocol (HTTP 1.0)	Informational	N/A	Version 1.0 of the application layer protocol used by web browsers talking to web servers over TCP; developed by the W3C rather than the IETF.
RFC 2045 RFC 2046 RFC 2047	Multipurpose Internet Mail Extensions	Draft standard	Elective	A means of encoding binary data and non-ASCII text for transmission through Internet email and other ASCII-oriented protocols.
RFC 2068	Hypertext Transfer Protocol (HTTP 1.1)	Proposed standard	Elective	Version 1.1 of the application layer protocol used by web browsers talking to web servers over TCP.
RFC 2141	Uniform Resource Names (URN) Syntax	Proposed standard	Elective	Similar to URLs but intended to refer to actual resources in a persistent fashion rather than the transient location of those resources.
RFC 2373	IP Version 6 Addressing Architecture	Proposed standard	Elective	The format and meaning of IPv6 addresses.
RFC 2396	Uniform Resource Identifiers (URI): Generic Syntax	Proposed standard	Elective	Similar to URLs but cut a broader path. For instance, ISBN numbers may be URIs even if the book cannot be retrieved over the Internet.
RFC 3501	Internet Message Access Protocol Version 4rev1	Proposed standard	Elective	A protocol for remotely accessing a mailbox stored on a server including downloading messages, deleting messages, and moving messages into and out of different folders.

The IETF has traditionally worked behind the scenes, codifying and standardizing existing practice. Although its activities are completely open to the public, it's traditionally been very low-profile. There simply aren't that many people who get excited about the details of network arcana like the Internet Gateway Message Protocol (IGMP). The participants in the process have mostly been engineers and computer scientists, including many from academia as well as the corporate world. Consequently, despite often vociferous debates about ideal implementations, most serious IETF efforts have produced reasonable standards.

Unfortunately, that can't be said of the IETF's efforts to produce web (as opposed to Internet) standards. In particular, the IETF's early effort to standardize HTML was a colossal failure. The refusal of Netscape and other key vendors to participate or even acknowledge the process was a crucial problem. That HTML was simple enough and high-profile enough to attract the attention of assorted market-droids and random flamers didn't help matters either. Thus, in October 1994 the World Wide Web Consortium was formed as a vendor-controlled body that might be able to avoid the pitfalls that plagued the IETF's efforts to standardize HTML and HTTP.

1.6.2 W3C Recommendations

Although the W3C standardization process is similar to the IETF process (a series of working drafts hashed out on mailing lists resulting in an eventual specification), the W3C is a fundamentally different organization. Whereas the IETF is open to participation by anyone, only corporations and other organizations may become members of the W3C. Individuals are specifically excluded. Furthermore, although nonprofit organizations like the World Wide Web Artists Consortium (WWWAC) may join the W3C, only the employees of these organizations may participate in W3C activities. Their volunteer members are not welcome. Specific individual experts are occasionally invited to participate on a particular working group even though they are not employees of a W3C member company. However, the number of such individuals is quite small relative to the number of interested experts in the broader community. Membership in the W3C costs $50,000 a year ($5,000 a year for nonprofits) with a minimum 3-year commitment. Membership in the IETF costs $0 a year with no commitment beyond a willingness to participate. And although many people participate in developing W3C standards, each standard is ultimately approved or vetoed by one individual, W3C director Tim Berners-Lee. IETF standards are approved by a consensus of the people who worked on the standard. Clearly, the IETF is a much more democratic (some would say anarchic) and open organization than the W3C.

Despite the W3C's strong bias toward the corporate members that pay its bills, it has so far managed to do a better job of navigating the politically tricky waters of Web standardization than the IETF. It has produced several HTML standards, as well as a variety of others such as HTTP, PICS, XML, CSS, MathML, and more. The W3C has had considerably less success in convincing vendors like Netscape and Microsoft to fully and consistently implement its standards.

The W3C has five basic levels of standards:

Note

A note is generally one of two things: either an unsolicited submission by a W3C member (similar to an IETF Internet draft) or random musings by W3C staff or related parties that do not actually describe a full proposal (similar to an IETF informational RFC). Notes will not necessarily lead to the formation of a working group or a W3C recommendation.

Working drafts

A working draft is a reflection of the current thinking of some (not necessarily all) members of a working group. It should eventually lead to a proposed recommendation, but by the time it does so it may have changed substantially.

Candidate recommendation

A candidate recommendation indicates that the working draft has reached consensus on all major issues and is ready for third-party comment and implementations. If the implementations do not uncover any obstructions, the spec can be promoted to a proposed recommendation.

Proposed recommendation

A proposed recommendation is mostly complete and unlikely to undergo more than minor editorial changes. The main purpose of a proposed recommendation is to work out bugs in the specification document rather than in the underlying technology being documented.

Recommendation

A recommendation is the highest level of W3C standard. However, the W3C is very careful not to actually call this a "standard" for fear of running afoul of antitrust statutes. The W3C describes a recommendation as a "work that represents consensus within W3C and has the Director's stamp of approval. W3C considers that the ideas or technology specified by a Recommendation are appropriate for widespread deployment and promote W3C's mission."

The W3C has not been around long enough to develop a need for a historical or informational standard status. Another difference the IETF and the W3C is that the W3C process rarely fails to elevate a standard to full recommendation status once work has actively commenced—that is, once a working group has been formed. This contrasts markedly with the IETF, which has more than a thousand proposed and draft standards, but only a few dozen actual standards.

PR Standards

In recent years, companies seeking a little free press or perhaps a temporary boost to their stock price have abused both the W3C and IETF standards processes. The IETF will accept a submission from anyone, and the W3C will accept a submission from any W3C member. The IETF calls these submissions "Internet drafts" and publishes them for six months before deleting them. The W3C refers to such submissions as "acknowledged submissions" and publishes them indefinitely. However, neither organization actually promises to do more than acknowledge receipt of these documents. In particular, they do not promise to form a working group or begin the standardization process. Nonetheless, press releases invariably misrepresent the submission of such a document as a far more significant event than it actually is. PR reps can generally count on suckering at least a few clueless reporters who aren't up to speed on the intimate details of the standardization process. However, you should recognize these ploys for what they are.

CHAPTER - 2

BASIC WEB CONCEPTS

Java can do a lot more than create flashy web pages. Nonetheless, many of your programs will be applets on web pages, servlets running on the server, or web services that need to talk to other web servers and clients. Therefore, it's important to have a solid understanding of the interaction between web servers and web browsers.

The Hypertext Transfer Protocol (HTTP) is a standard that defines how a web client talks to a server and how data is transferred from the server back to the client. The architecture and design of the HTTP protocol is Representational State Transfer (REST). HTTP can be used to transfer data in essentially any format, from TIFF pictures to Microsoft Word documents to DBase files. However, far and away the most common format for data transferred over the Web and in some sense the Web's native format is the Hypertext Markup Language (HTML). HTML is a simple standard for describing the semantic value of textual data. You can say "this is a header", "this is a list item", "this deserves emphasis", and so on, but you can't specify how headers, lists, and other items are formatted: formatting is up to the browser. HTML is a "hypertext markup language" because it includes a way to specify links to other documents identified by URLs. A URL is a way to unambiguously identify the location of a resource on the Internet. To understand network programming, you'll need to understand URLs, HTML, and HTTP in somewhat more detail than the average web page designer.

2.1 URIs

A Uniform Resource Identifier (URI) is a string of characters in a particular syntax that identifies a resource. The resource identified may be a file on a server, but it may also be an email address, a news message, a book, a person's name, an Internet host, the current stock price of Sun Microsystems, or something else. An absolute URI is made up of a scheme for the URI and a scheme-specific part, separated by a colon, like this:

```
scheme:scheme-specific-part
```

The syntax of the scheme-specific part depends on the scheme being used. Current schemes include:

data

Base64-encoded data included directly in a link; see RFC 2397

file

A file on a local disk

ftp

An FTP server

http

A World Wide Web server using the Hypertext Transfer Protocol

gopher

A Gopher server

mailto

An email address

news

A Usenet newsgroup

telnet

A connection to a Telnet-based service

urn

A Uniform Resource Name

In addition, Java makes heavy use of nonstandard custom schemes such as rmi, jndi, and doc for various purposes. We'll look at the mechanism behind this in Chapter 16, when we discuss protocol handlers.

There is no specific syntax that applies to the scheme-specific parts of all URIs. However, many have a hierarchical form, like this:

```
//authority/path?query
```

The authority part of the URI names the authority responsible for resolvin g the rest of the URI. For instance, the URI http://www.ietf.org/rfc/rfc2396.txt has the scheme http and the authority www.ietf.org. This means the server at www.ietf.org is responsible for mapping the path /rfc/rfc2396.txt to a resource. This URI does not have a query part. The URI http://www.powells.com/cgi-bin/biblio?inkey=62-1565928709-0 has the scheme http, the authority www.powells.com, the path /biblio, and the query inkey=62-1565928709-0. The URI urn:isbn:156592870 has the scheme urn but doesn't follow the hierarchical //authority/path?query form for scheme-specific parts.

Although most current examples of URIs use an Internet host as an authority, future schemes may not. However, if the authority is an Internet host, optional usernames and ports may also be provided to make the authority more specific. For example, the URI ftp://mp3:mp3@ci43198-a.ashvil1.nc.home.com:33/VanHalen-Jump.mp3 has the authority mp3:mp3@ci43198-a.ashvil1.nc.home.com:33. This authority has the username mp3, the password mp3, the host ci43198-a.ashvil1.nc.home.com, and the port 33. It has the scheme ftp and the path /VanHalen-Jump.mp3. (In most cases, including the password in the URI is a big security hole unless, as here, you really do want everyone in the universe to know the password.)

The path (which includes its initial /) is a string that the authority can use to determine which resource is identified. Different authorities may interpret the same path to refer to different resources. For instance, the path /index.html means one thing when the authority is www.landoverbaptist.org and something very different when the authority is www.churchofsatan.com. The path may be hierarchical, in which case the individual parts are separated by forward slashes, and the . and .. operators are used to navigate the hierarchy. These are derived from the pathname syntax on the Unix operating systems where the Web and URLs were invented. They conveniently map to a filesystem stored on a Unix web server. However, there is no guarantee that the components of any particular path actually correspond to files or directories on any particular filesystem. For example, in the URI http://www.amazon.com/exec/obidos/ISBN%3D1565924851/cafeaulaitA/002-3777605-3043449, all the pieces of the hierarchy are just used to pull information out of a database that's never stored in a filesystem. ISBN%3D1565924851 selects the particular book from the database by its ISBN number, cafeaulaitA specifies who gets the referral fee if a purchase is made from this link, and 002-3777605-3043449 is a session key used to track the visitor's path through the site.

Some URIs aren't at all hierarchical, at least in the filesystem sense. For example, snews://secnews.netscape.com/netscape.devs-java has a path of /netscape.devs-java. Although there's some hierarchy to the newsgroup names indicated by the . between netscape and netscape.devs-java, it's not visible as part of the URI.

The scheme part is composed of lowercase letters, digits, and the plus sign, period, and hyphen. The other three parts of a typical URI (authority, path, and query) should each be composed of the ASCII alphanumeric characters; that is, the letters A-Z, a-z, and the digits 0-9. In addition, the punctuation characters - _ . ! ~ * ' may also be used. All other characters, including non-ASCII alphanumerics such as á and , should be escaped by a percent sign (%) followed by the hexadecimal code for the character. For instance, á would be encoded as %E1. A URL so transformed is said to have been "x-www-form-urlencoded".

This process assumes that the character set is the Latin 1. The URI and URL specifications don't actually say what character set should be used, which means most software tends to use the local default character set. Thus, URLs containing non-ASCII characters aren't very interoperable across different platforms and languages. With the Web becoming more international and less English daily, this situation has become increasingly problematic. Work is ongoing to define Internationalized Resource Identifiers (IRIs) that can use the full range of Unicode. At the time of this writing, the IRI draft specification indicates that non-ASCII characters should be encoded by first converting them to UTF-8, then percent-escaping each byte of the UTF-8, as specified above. For instance, the Greek letter is Unicode code point 3C0. In UTF-8, this letter is encoded as the three bytes E0, A7, 80. Thus in a URL it would be encoded as %E0%A7%80.

Punctuation characters such as / and @ must also be encoded with percent escapes if they are used in any role other than what's specified for them in the scheme-specific part of a particular URL. For example, the forward slashes in the URI http://www.cafeaulait.org/books/javaio/ do not need to be encoded as %2F because they serve to delimit the hierarchy as specified for the http URI scheme. However, if a filename includes a / character—for instance, if the last directory were named Java I/O instead of javaio to more closely match the name of the book—the URI would have to be written as http://www.cafeaulait.org/books/Java%20I%2FO/. This is not as farfetched as it might sound to Unix or Windows users. Mac filenames frequently include a forward slash. Filenames on many platforms often contain characters that need to be encoded, including @, $, +, =, and many more.

2.1.1 URNs

There are two types of URIs: Uniform Resource Locators (URLs) and Uniform Resource Names (URNs). A URL is a pointer to a particular resource on the Internet at a particular location. For example, http://www.oreilly.com/catalog/javanp3/ is one of several URLs for the book Java Network Programming. A URN is a name for a particular resource but without reference to a particular location. For instance, urn:isbn:1565928709 is a URN referring to the same book. As this example shows, URNs, unlike URLs, are not limited to Internet resources.

The goal of URNs is to handle resources that are mirrored in many different locations or that have moved from one site to another; they identify the resource itself, not the place where the resource lives. For instance, when given a URN for a particular piece of software, an FTP program should get the file from the nearest mirror site. Given a URN for a book, a browser might reserve the book at the local library or order a copy from a bookstore.

A URN has the general form:

```
urn:namespace:resource_name
```

The *namespace* is the name of a collection of certain kinds of resources maintained by some authority. The *resource_name* is the name of a resource within that collection. For instance, the URN urn:ISBN:1565924851 identifies a resource in the ISBN namespace with the identifier 1565924851. Of all the books published, this one selects the first edition of Java I/O.

The exact syntax of resource names depends on the namespace. The ISBN namespace expects to see strings composed of 10 or 13 characters, all of which are digits—with the single exception that the last character may be the letter X (either upper- or lowercase) instead. Furthermore, ISBNs may contain hyphens that are ignored when comparing. Other namespaces will use very different syntaxes for resource names. The IANA is responsible for handing out namespaces to different organizations, as described in RFC 3406. Basically, you have to submit an Internet draft to the IETF and publish an announcement on the urn-nid mailing list for public comment and discussion before formal standardization.

2.1.2 URLs

A URL identifies the location of a resource on the Internet. It specifies the protocol used to access a server (e.g., FTP, HTTP), the name of the server, and the location of a file on that server. A typical URL looks like http://www.ibiblio.org/javafaq/javatutorial.html. This specifies that there is a file called javatutorial.html in a directory called javafaq on the server www.ibiblio.org, and that this file can be accessed via the HTTP protocol. The syntax of a URL is:

```
protocol://username@hostname:port/path/filename?query#fragment
```

Here the protocol is another word for what was called the scheme of the URI. (Scheme is the word used in the URI RFC. Protocol is the word used in the Java documentation.) In a URL, the protocol part can be file, ftp, http, https, gopher, news, telnet, wais, or various other strings (though not urn).

The hostname part of a URL is the name of the server that provides the resource you want, such as www.oreilly.com or utopia.poly.edu. It can also be the server's IP address, such as 204.148.40.9 or 128.238.3.21. The username is an optional username for the server. The port number is also optional. It's not necessary if the service is running on its default port (port 80 for HTTP servers).

The path points to a particular directory on the specified server. The path is relative to the document root of the server, not necessarily to the root of the filesystem on the server. As a rule, servers that are open to the public do not show their entire filesystem to clients. Rather, they show only the contents of a specified directory. This directory is called the document root, and all paths and filenames are relative to it. Thus, on a Unix server, all files that are available to the public might be in /var/public/html, but to somebody connecting from a remote machine, this directory looks like the root of the filesystem.

The filename points to a particular file in the directory specified by the path. It is often omitted—in which case, it is left to the server's discretion what file, if any, to send. Many servers send an index file for that directory, often called index.html or Welcome.html. Some send a list of the files and folders in the directory, as shown in Figure 2-1. Others may send a 403 Forbidden error message, as shown in Figure 2-2.

Figure 2-1. A web server configured to send a directory list when no index file exists

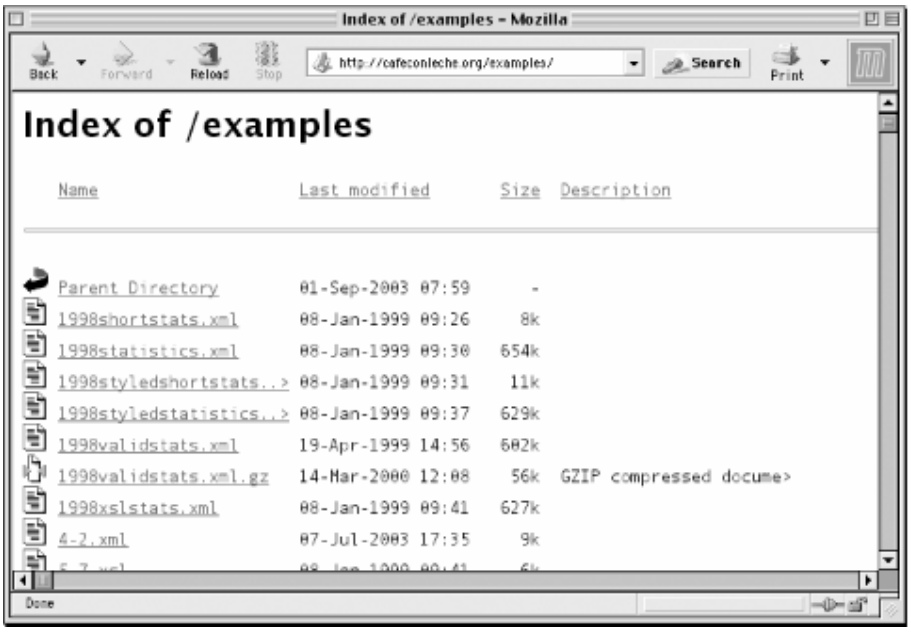
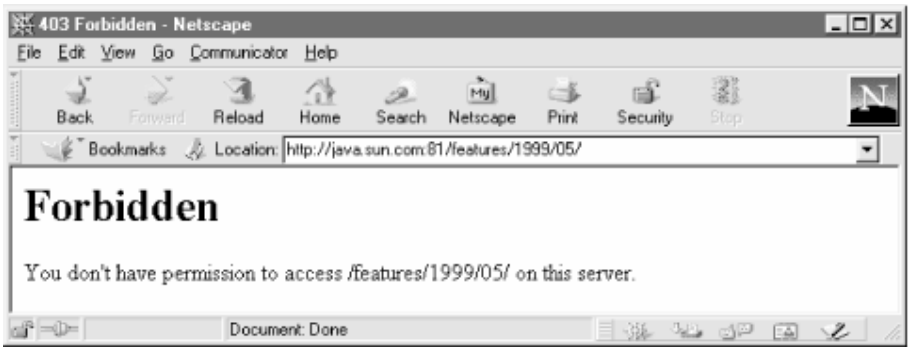


Figure 2-2. A web server configured to send a 403 error when no index file exists



The query string provides additional arguments for the server. It's commonly used only in http URLs, where it contains form data for input to programs running on the server.

Finally, the fragment references a particular part of the remote resource. If the remote resource is HTML, the fragment identifier names an anchor in the HTML document. If the remote resource is XML, the fragment identifier is an XPointer. Some documents refer to the fragment part of the URL as a "section"; Java documents rather unaccountably refer to the fragment identifier as a "Ref". A named anchor is created in an HTML document with a tag, like this:

```
<A NAME="xtocid1902914">Comments</A>
```

This tag identifies a particular point in a document. To refer to this point, a URL includes not only the document's filename but the named anchor separated from the rest of the URL by a #:

```
http://www.cafeaulait.org/javafaq.html#xtocid1902914
```

 Technically, a string that contains a fragment identifier is a URL reference, not a URL. Java, however, does not distinguish between URLs and URL references.

2.1.3 Relative URLs

A URL tells the web browser a lot about a document: the protocol used to retrieve the document, the name of the host where the document lives, and the path to that document on the host. Most of this information is likely to be the same for other URLs that are referenced in the document. Therefore, rather than requiring each URL to be specified in its entirety, a URL may inherit the protocol, hostname, and path of its parent document (i.e., the document in which it appears). URLs that aren't complete but inherit pieces from their parent are called relative URLs. In contrast, a completely specified URL is called an absolute URL. In a relative URL, any pieces that are missing are assumed to be the same as the corresponding pieces from the URL of the document in which the URL is found. For example, suppose that while browsing http://www.ibiblio.org/javafaq/javatutorial.html you click on this hyperlink:

```
<a href="javafaq.html">
```

The browser cuts javatutorial.html off the end of http://www.ibiblio.org/javafaq/javatutorial.html to get http://www.ibiblio.org/javafaq/. Then it attaches javafaq.html onto the end of http://www.ibiblio.org/javafaq/ to get http://www.ibiblio.org/javafaq/javafaq.html. Finally, it loads that document.

If the relative link begins with a `/`, then it is relative to the document root instead of relative to the current file. Thus, if you click on the following link while browsing http://www.ibiblio.org/javafaq/javatutorial.html:

```
<a href="/boutell/faq/www_faq.html">
```

the browser would throw away /javafaq/javatutorial.html and attach /boutell/faq/www_faq.html to the end of http://www.ibiblio.org to get http://www.ibiblio.org/boutell/faq/www_faq.html.

Relative URLs have a number of advantages. First—and least important—they save a little typing. More importantly, relative URLs allow a single document tree to be served by multiple protocols: for instance, both FTP and HTTP. The HTTP might be used for direct surfing, while the FTP could be used for mirroring the site. Most importantly of all, relative URLs allow entire trees of documents to be moved or copied from one site to another without breaking all the internal links.

2.2 HTML, SGML, and XML

HTML is the primary format used for Web documents. As I said earlier, HTML is a simple standard for describing the semantic content of textual data. The idea of describing a text's semantics rather than its appearance comes from an older standard called the Standard Generalized Markup Language (SGML). Standard HTML is an instance of SGML. SGML was invented in the mid-1970s by Charles Goldfarb, Edward Mosher, and Raymond Lorie at IBM. SGML is now an International Standards Organization (ISO) standard, specifically ISO 8879:1986.

SGML and, by inheritance, HTML are based on the notion of design by meaning rather than design by appearance. You don't say that you want some text printed in 18-point type; you say that it is a top-level heading (`<H1>` in HTML). Likewise, you don't say that a word should be placed in italics. Rather, you say it should be emphasized (`<EM>` in HTML). It is left to the browser to determine how to best display headings or emphasized text.

The tags used to mark up the text are case-insensitive. Thus, `<STRONG>` is the same as `<strong>` is the same as `<Strong>` is the same as `<StrONg>`. Some tags have a matching end-tag to define a region of text. An end-tag is the same as the start-tag, except that the opening angle bracket is followed by a `/`. For example: `<STRONG>this text is strong</STRONG>`; `<EM>this text is emphasized</EM>`. The entire text from the beginning of the start-tag to the end of the end-tag is called an element. Thus, `<STRONG>this text is strong</STRONG>` is a `STRONG` element.

HTML elements may nest but they should not overlap. The first line in the following example is standard-conforming. The second line is not, though many browsers accept it nonetheless:

```
<STRONG><EM>Jack and Jill went up the hill</EM></STRONG>

<STRONG><EM>to fetch a pail of water</STRONG></EM>
```

Some elements have additional attributes that are encoded as name-value pairs on the start-tag. The `<H1>` tag and most other paragraph-level tags may have an `ALIGN` attribute that says whether the header should be centered, left-aligned, or right-aligned. For example:

```
<H1 ALIGN=CENTER> This is a centered H1 heading </H1>
```

The value of an attribute may be enclosed in double or single quotes, like this:

```
<H1 ALIGN="CENTER"> This is a centered H1 heading </H1>

<H2 ALIGN='LEFT'> This is a left-aligned H2 heading
</H2>
```

Quotes are required only if the value contains embedded spaces. When processing HTML, you need to be prepared for attribute values that do and don't have quotes.

There have been several versions of HTML over the years. The current standard is HTML 4.0, most of which is supported by current web browsers, with occasional exceptions. Furthermore, several companies, notably Netscape, Microsoft, and Sun, have added nonstandard extensions to HTML. These include blinking text, inline movies, frames, and, most importantly for this book, applets. Some of these extensions—for example, the `<APPLET>` tag—are allowed but deprecated in HTML 4.0. Others, such as Netscape's notorious `<BLINK>`, come out of left field and have no place in a semantically-oriented language like HTML.

HTML 4.0 may be the end of the line, aside from minor fixes. The W3C has decreed that HTML is getting too bulky to layer more features on top of. Instead, new development will focus on XML, a semantic language that allows page authors to create the elements they need rather than relying on a few fixed elements such as `P` and `LI`. For example, if you're writing a web page with a price list, you would likely have an `SKU` element, a `PRICE` element, a `MANUFACTURER` element, a `PRODUCT` element, and so forth. That might look something like this:

```
<PRODUCT MANUFACTURER="IBM">
  <NAME>Lotus Smart Suite</NAME>
  <VERSION>9.8</VERSION>
  <PLATFORM>Windows</PLATFORM>
  <PRICE CURRENCY="US">299.95</PRICE>
  <SKU>D05WGML</SKU>
</PRODUCT>
```

This looks a lot like HTML, in much the same way that Java looks like C. There are elements and attributes. Tags are set off by `<` and `>`. Attributes are enclosed in quotation marks, and so forth. However, instead of being limited to a finite set of tags, you can create all the new and unique tags you need. Since no browser can know in advance all the different elements that may appear, a stylesheet is used to describe how each of the items should be displayed.

XML has another advantage over HTML that may not be obvious from this simple example. HTML can be quite sloppy. Elements are opened but not closed. Attribute values may or may not be enclosed in quotes. The quotes may or may not be present. XML tightens all this up. It lays out very strict requirements for the syntax of a well-formed XML document, and it requires that browsers reject all malformed documents. Browsers may not attempt to fix the problem and make a best-faith effort to display what they think the author meant. They must simply report the error. Furthermore, an XML document may have a Document Type Definition (DTD), which can impose additional constraints on valid documents. For example, a DTD may require that every `PRODUCT` element contain exactly one `NAME` element. This has a number of advantages, but the key one here is that XML documents are far easier to parse than HTML documents. As a programmer, you will find it much easier to work with XML than HTML.

XML can be used both for pure XML pages and for embedding new kinds of content in HTML and XHTML. For example, the Mathematical Markup Language, MathML, is an XML application for including mathematical equations in web pages. SMIL, the Synchronized Multimedia Integration Language, is an XML application for including timed multimedia such as slide shows and subtitled videos on web pages. More recently, the W3C has released several versions of XHTML. This language uses the familiar HTML vocabulary (`p` for paragraphs, `tr` for table rows, `img` for pictures, and so forth) but requires the document to follow XML's stricter rules: all attribute values must be quoted; every start-tag must have a matching end-tag; elements can nest but cannot overlap; etc. For a lot more information about XML, see XML in a Nutshell by Elliotte Rusty Harold and W. Scott Means (O'Reilly).

2.3 HTTP

HTTP is the standard protocol for communication between web browsers and web servers. HTTP specifies how a client and server establish a connection, how the client requests data from the server, how the server responds to that request, and finally, how the connection is closed. HTTP connections use the TCP/IP protocol for data transfer. For each request from client to server, there is a sequence of four steps:

Making the connection

The client establishes a TCP connection to the server on port 80, by default; other ports may be specified in the URL.

Making a request

The client sends a message to the server requesting the page at a specified URL. The format of this request is typically something like:

```
GET /index.html HTTP/1.0
```

`GET` specifies the operation being requested. The operation requested here is for the server to return a representation of a resource. `/index.html` is a relative URL that identifies the resource requested from the server. This resource is assumed to reside on the machine that receives the request, so there is no need to prefix it with `http://www.thismachine.com/`.`HTTP/1.0` is the version of the protocol that the client understands. The request is terminated with two carriage return/linefeed pairs (`\r\n\r\n` in Java parlance), regardless of how lines are terminated on the client or server platform.

Although the `GET` line is all that is required, a client request can include other information as well. This takes the following form:

`Keyword``: `*`Value`*

The most common such keyword is `Accept`, which tells the server what kinds of data the client can handle (though servers often ignore this). For example, the following line says that the client can handle four MIME media types, corresponding to HTML documents, plain text, and JPEG and GIF images:

```
Accept: text/html, text/plain, image/gif, image/jpeg
```

`User-Agent` is another common keyword that lets the server know what browser is being used, allowing the server to send files optimized for the particular browser type. The line below says that the request comes from Version 2.4 of the Lynx browser:

```
User-Agent: Lynx/2.4 libwww/2.1.4
```

All but the oldest first-generation browsers also include a `Host` field specifying the server's name, which allows web servers to distinguish between different named hosts served from the same IP address. Here's an example:

```
Host: www.cafeaulait.org
```

Finally, the request is terminated with a blank line—that is, two carriage return/linefeed pairs, `\r\n\r\n`. A complete request might look like this:

```
GET /index.html HTTP/1.0

Accept: text/html, text/plain, image/gif, image/jpeg

User-Agent: Lynx/2.4 libwww/2.1.4

Host: www.cafeaulait.org
```

In addition to `GET`, there are several other request types. `HEAD` retrieves only the header for the file, not the actual data. This is commonly used to check the modification date of a file, to see whether a copy stored in the local cache is still valid. `POST` sends form data to the server, `PUT` uploads a resource to the server, and `DELETE` removes a resource from the server.

The response

The server sends a response to the client. The response begins with a response code, followed by a header full of metadata, a blank line, and the requested document or an error message. Assuming the requested document is found, a typical response looks like this:

```
HTTP/1.1 200 OK
Date: Mon, 15 Sep 2003 21:06:50 GMT
Server: Apache/2.0.40 (Red Hat Linux)
Last-Modified: Tue, 15 Apr 2003 17:28:57 GMT
Connection: close
Content-Type: text/html; charset=ISO-8859-1
Content-length: 107

<html>
<head>
<title>
A Sample HTML file
</title>
</head>
<body>
The rest of the document goes here
</body>
</html>
```

The first line indicates the protocol the server is using (`HTTP/1.1`), followed by a response code. `200 OK` is the most common response code, indicating that the request was successful. Table 2-1 is a complete list of the response codes used by HTTP 1.0; HTTP 1.1 adds many more to this list. The other header lines identify the date the request was made in the server's time frame, the server software (Apache 2.0.40), the date this document was last modified, a promise that the server will close the connection when it's finished sending, the MIME content type, and the length of the document delivered (not counting this header)—in this case, 107 bytes.

Closing the connection

Either the client or the server or both close the connection. Thus, a separate network connection is used for each request. If the client reconnects, the server retains no memory of the previous connection or its results. A protocol that retains no memory of past requests is called stateless; in contrast, a stateful protocol such as FTP can process many requests before the connection is closed. The lack of state is both a strength and a weakness of HTTP.

Table 2-1. HTTP 1.0 response codes

Response code	Meaning
2xx Successful	Response codes between 200 and 299 indicate that the request was received, understood, and accepted.
200 OK	This is the most common response code. If the request used `GET` or `POST`, the requested data is contained in the response along with the usual headers. If the request used `HEAD`, only the header information is included.
201 Created	The server has created a data file at a URL specified in the body of the response. The web browser should now attempt to load that URL. This is sent only in response to `POST` requests.
202 Accepted	This rather uncommon response indicates that a request (generally from `POST`) is being processed, but the processing is not yet complete so no response can be returned. The server should return an HTML page that explains the situation to the user, provides an estimate of when the request is likely to be completed, and, ideally, has a link to a status monitor of some kind.
204 No Content	The server has successfully processed the request but has no information to send back to the client. This is usually the result of a poorly written form-processing program that accepts data but does not return a response to the user indicating that it has finished.
3xx Redirection	Response codes from 300 to 399 indicate that the web browser needs to go to a different page.
300 Multiple Choices	The page requested is available from one or more locations. The body of the response includes a list of locations from which the user or web browser can pick the most appropriate one. If the server prefers one of these locations, the URL of this choice is included in a `Location` header, which web browsers can use to load the preferred page.
301 Moved Permanently	The page has moved to a new URL. The web browser should automatically load the page at this URL and update any bookmarks that point to the old URL.
302 Moved Temporarily	This unusual response code indicates that a page is temporarily at a new URL but that the document's location will change again in the foreseeable future, so bookmarks should not be updated.
304 Not	The client has performed a `GET` request but used the `If-Modified-`

Response code	Meaning
Modified	`Since` header to indicate that it wants the document only if it has been recently updated. This status code is returned because the document has not been updated. The web browser will now load the page from a cache.
4xx Client Error	Response codes from 400 to 499 indicate that the client has erred in some fashion, although the error may as easily be the result of an unreliable network connection as of a buggy or nonconforming web browser. The browser should stop sending data to the server as soon as it receives a 4xx response. Unless it is responding to a `HEAD` request, the server should explain the error status in the body of its response.
400 Bad Request	The client request to the server used improper syntax. This is rather unusual, although it is likely to happen if you're writing and debugging a client.
401 Unauthorized	Authorization, generally username and password controlled, is required to access this page. Either the username and password have not yet been presented or the username and password are invalid.
403 Forbidden	The server understood the request but is deliberately refusing to process it. Authorization will not help. One reason this occurs is that the client asks for a directory listing but the server is not configured to provide it, as shown in Figure 2-1.
404 Not Found	This most common error response indicates that the server cannot find the requested page. It may indicate a bad link, a page that has moved with no forwarding address, a mistyped URL, or something similar.
5xx Server Error	Response codes from 500 to 599 indicate that something has gone wrong with the server, and the server cannot fix the problem.
500 Internal Server Error	An unexpected condition occurred that the server does not know how to handle.
501 Not Implemented	The server does not have the feature that is needed to fulfill this request. A server that cannot handle `POST` requests might send this response to a client that tried to `POST` form data to it.
502 Bad Gateway	This response is applicable only to servers that act as proxies or gateways. It indicates that the proxy received an invalid response from a server it was connecting to in an effort to fulfill the request.
503 Service Unavailable	The server is temporarily unable to handle the request, perhaps as a result of overloading or maintenance.

HTTP 1.1 more than doubles the number of responses. However, a response code from 200 to 299 always indicates success, a response code from 300 to 399 always indicates redirection, one from 400 to 499 always indicates a client error, and one from 500 to 599 indicates a server error.

HTTP 1.0 is documented in the informational RFC 1945; it is not an official Internet standard because it was primarily developed outside the IETF by early browser and server vendors. HTTP 1.1 is a proposed standard being developed by the W3C and the HTTP working group of the IETF. It provides for much more flexible and powerful communication between the client and the server. It's also a lot more scalable. It's documented in RFC 2616. HTTP 1.0 is the basic version of the protocol. All current web servers and browsers understand it. HTTP 1.1 adds numerous features to HTTP 1.0, but doesn't change the underlying design or architecture in any significant way. For the purposes of this book, it will usually be sufficient to understand HTTP 1.0.

The primary improvement in HTTP 1.1 is connection reuse. HTTP 1.0 opens a new connection for every request. In practice, the time taken to open and close all the connections in a typical web session can outweigh the time taken to transmit the data, especially for sessions with many small documents. HTTP 1.1 allows a browser to send many different requests over a single connection; the connection remains open until it is explicitly closed. The requests and responses are all asynchronous. A browser doesn't need to wait for a response to its first request before sending a second or a third. However, it remains tied to the basic pattern of a client request followed by a server response. Each request and response has the same basic form: a header line, an HTTP header containing metadata, a blank line, and then the data itself.

There are a lot of other, smaller improvements in HTTP 1.1. Requests include a `Host` header field so that one web server can easily serve different sites at different URLs. Servers and browsers can exchange compressed files and particular byte ranges of a document, both of which decrease network traffic. And HTTP 1.1 is designed to work much better with proxy servers. HTTP 1.1 is a superset of HTTP 1.0, so HTTP 1.1 web servers have no trouble interacting with older browsers that only speak HTTP 1.0, and vice versa.

2.4 MIME Media Types

MIME is an open standard for sending multipart, multimedia data through Internet email. The data may be binary, or it may use multiple ASCII and non-ASCII character sets. Although MIME was originally intended just for email, it has become a widely used technique to describe a file's contents so that client software can tell the difference between different kinds of data. For example, a web browser uses MIME to tell whether a file is a GIF image or a printable PostScript file.

 Officially, MIME stands for Multipurpose Internet Mail Extensions, which is the expansion of the acronym used in RFC 2045. However, you will hear other versions—most frequently Multipart Internet Mail Extensions and Multimedia Internet Mail Extensions.

MIME supports more than 100 predefined types of content. Content types are classified at two levels: a type and a subtype. The type shows very generally what kind of data is contained: is it a picture, text, or movie? The subtype identifies the specific type of data: GIF image, JPEG image, TIFF image. For example, HTML's content type is `text/html`; the type is `text`, and the subtype is `html`. The content type for a GIF image is `image/gif`; the type is `image`, and the subtype is `gif`. Table 2-2 lists the more common defined content types. On most systems, a simple text file maintains a mapping between MIME types and the application used to process that type of data; on Unix, this file is called mime.types. The most current list of registered MIME types is available from http://www.iana.org/assignments/media-types/. For more on MIME, see the comp.mail.mime FAQ at http://www.uni-giessen.de/faq/archiv/mail.mime-faq.part1-9/.

Web servers use MIME to identify the kind of data they're sending. Web clients use MIME to identify the kind of data they're willing to accept. Most web servers and clients understand at least two MIME text content types, `text/html` and `text/plain`, and two image formats, `image/gif` and `image/jpeg`. More recent browsers also understand `application/xml` and several other image formats. Java relies on MIME types to pick the appropriate content handler for a particular stream of data.

Table 2-2. Predefined MIME content types

Type	Subtype	Description
text		The document represents printable text.
	calendar	Calendaring and scheduling information in the iCalendar format; see RFC 2445.
	css	A Cascading Style Sheet used for HTML and XML.
	directory	Address book information such as name, phone number, and email address; used by Netscape vCards; defined in RFCs 2425 and 2426.
	enriched	A very simple HTML-like language for adding basic font and paragraph-level formatting such as bold and italic to email; used by Eudora; defined in RFC 1896.
	html	Hypertext Markup Language as used by web browsers.
	plain	This is supposed to imply raw ASCII text. However, some web servers use `text/plain` as the default MIME type for any file they can't recognize. Therefore, anything and everything, most notably *.class* byte code files, can get identified as a `text/plain` file.
	richtext	An HTML-like markup for encoding formatting into pure ASCII text. It's never really caught on, in large part because of the popularity of HTML.
	rtf	An incompletely defined Microsoft format for word processing files.
	sgml	The Standard Generalized Markup Language; ISO standard

Type	Subtype	Description
		8879:1986.
	tab-separated-values	The interchange format used by many spreadsheets and databases; records are separated by linebreaks and fields by tabs.
	xml	The W3C standard Extensible Markup Language. For various technical reasons, `application/xml` should be used instead, but often isn't.
multipart		Multipart MIME messages encode several different files into one message.
	mixed	Several message parts intended for sequential viewing.
	alternative	The same message in multiple formats so a client may choose the most convenient one.
	digest	A popular format for merging many email messages into a single digest; used by many mailing lists and some FAQ lists.
	parallel	Several parts intended for simultaneous viewing.
	byteranges	Several separately contiguous byte ranges; used in HTTP 1.1.
	encrypted	One part for the body of the message and one part for the information necessary to decode the message.
	signed	One part for the body of the message and one part for the digital signature.
	related	Compound documents formed by aggregating several smaller parts.
	form-data	Form responses.
message		An email message.
	external-body	Just the headers of the email message; the message's body is not included but exists at some other location and is referenced, perhaps by a URL.
	http	An HTTP 1.1 request from a web client to a web server.
	news	A news article.
	partial	Part of a longer email message that has been split into multiple parts to allow transmission through email gateways.
	rfc822	A standard email message including headers.
image		Two-dimensional pictures.
	cgm	A Computer Graphics Metafile format image. CGM is ISO standard 8632:1992 for device-independent vector graphics and bitmap images.
	g3fax	The standard for bitmapped fax images.

Type	Subtype	Description
	gif	A Graphics Interchange Format image.
	jpeg	The Joint Photographic Experts Group file format for bitmapped images with lossy compression.
	png	A Portable Network Graphics Format image. The format was developed at the W3C as a modern replacement for GIF that supports 24-bit color and is not encumbered by patents.
	tiff	The Tagged Image File format from Adobe.
audio		Sound.
	basic	8-bit ISDN -law encoded audio with a single channel and a sample rate of eight kilohertz. This is the format used by .au and *.snd* files and supported by the `java.applet.AudioClip` class.
video		Video.
	mpeg	The Motion Picture Experts Group format for video data with lossy compression.
	quicktime	Apple's proprietary QuickTime movie format. Before being included in a MIME message, QuickTime files must be "flattened".
model		3-D images.
	vrml	A Virtual Reality Modeling Language file, a format for 3-D data on the Web.
	iges	The Initial Graphics Exchange Specification for interchanging documents between different CAD programs.
	mesh	The mesh structures used in finite element and finite difference methods.
application		Binary data specific to some application.
	octet-stream	Unspecified binary data, which is usually saved into a file for the user. This MIME type is sometimes used to serve *.class* byte code files.
	java	A nonstandard subtype sometimes used to serve *.class* byte code files.
	postscript	Adobe PostScript.
	dca-rft	IBM's Document Content Architecture-Richly Formatted Text.
	mac-BinHex40	A means of encoding the two forks of a Macintosh document in a single ASCII file.
	pdf	An Adobe Acrobat file.
	zip	A zip compressed file.
	macwriteii	A MacWrite II word-processing document.

Type	Subtype	Description
	msword	A Microsoft Word document.
	xml+xhtml	An XHTML document
	xml	An Extensible Markup Language document.

A MIME-compliant program is not required to understand all these different types of data; it just needs to recognize what it can and cannot handle. Many programs—Netscape Navigator, for example—use various helper programs to display types of content they themselves don't understand.

MIME allows you to define additional nonstandard subtypes by using the prefix `x-`. For example, the content type `application/x-tex` has the MIME type `application` and the nonstandard subtype `x-tex` for a TeX document. These x-types are not guaranteed to be understood by any program other than the one that created them. Indeed, two programs may use the same x-type to mean two completely different things, or different programs may use different x-types to mean the same thing. However, many nonstandard types have come into common use; some of the more common ones are listed in Table 2-3.

Table 2-3. X-types

Type	X-subtype	Description
application		Subtypes of an application; the name of the subtype is usually a file format name or an application name.
	x-aiff	SGI's AIFF audio data format.
	x-bitmap	An X Windows bitmap image.
	x-gzip	Data compressed in the GNU gzip format.
	x-dvi	A TeX DVI document.
	x-framemaker	A FrameMaker document.
	x-latex	A LaTeX document.
	x-macBinHex40	Identical to `application/mac-BinHex40`, but older software may use this x-type instead.
	x-mif	A FrameMaker MIF document.
	x-sd	A session directory protocol announcement, used to announce MBONE events.
	x-shar	A shell archive; the Unix equivalent of a Windows or Macintosh self-extracting archive. Software shouldn't be configured to unpack shell archives automatically, because a shell archive can call any program the user who runs it has the rights to call.

Type	X-subtype	Description
	x-tar	A tar archive.
	x-gtar	A GNU tar archive.
	x-tcl	A tool command language (TCL) program. You should never configure your web browser or email program to automatically run programs you download from the web or receive in email messages.
	x-tex	A TeX document.
	x-texinfo	A GNU texinfo document.
	x-troff	A troff document.
	x-troff-man	A troff document written with the *man* macros.
	x-troff-me	A troff document that should be processed using the *me* macros.
	x-troff-ms	A troff document that should be processed using the *ms* macros.
	x-wais-source	A WAIS source.
	x-www-form-urlencoded	A string that has been encoded like a URL, with + replacing spaces and % escapes replacing non-alphanumeric characters that aren't separators.
audio		
	x-aiff	The same as `application/x-aiff`: an AIFF audio file.
	x-mpeg	The MP3 sound format.
	x-mpeg.mp3	The MP3 sound format.
	x-wav	The Windows WAV sound format.
image		
	x-fits	The FITS image format used primarily by astronomers.
	x-macpict	A Macintosh PICT image.
	x-pict	A Macintosh PICT image.
	x-macpaint	A MacPaint image.
	x-pbm	A portable bitmap image.
	x-portable-bitmap	A portable bitmap image.
	x-pgm	A PGM image.
video		
	x-msvideo	A Microsoft AVI Video for Windows.

Type	X-subtype	Description
	x-sgi-movie	A Silicon Graphics movie.

2.5 Server-Side Programs

These days many web pages are not served from static files on the hard drive. Instead, the server generates them dynamically to meet user requests. The content may be pulled from a database or generated algorithmically by a program. Indeed, the actual page delivered to the client may contain data combined from several different sources. In Java, such server-side programs are often written using servlets or Java Server Pages (JSP). They can also be written with other languages, such as C and Perl, or other frameworks, such as ASP and PHP. The concern in this book is not so much with how these programs are written as with how your programs communicate with them. One advantage to HTTP is that it really doesn't matter how the other side of the connection is written, as long as it speaks the same basic HTTP protocol.

The simplest server-side programs run without any input from the user. From the viewpoint of the client, these programs are accessed like any other web page and aren't of much concern to this book. The difference between a web page produced by a program that takes no input and a web page written in static HTML is all on the server side. When writing clients, you don't need to know or care whether the web server is feeding you a file or the output of some program it ran. Your interface to the server is the same in either case.

A slightly more complex server-side program processes user input from HTML forms. A web form is essentially just a way of collecting input from the user, dividing it into neat pieces, and passing those pieces to some program on the server. A client written in Java can perform the same function, either by asking the user for input in its own GUI or by providing its own unique information.

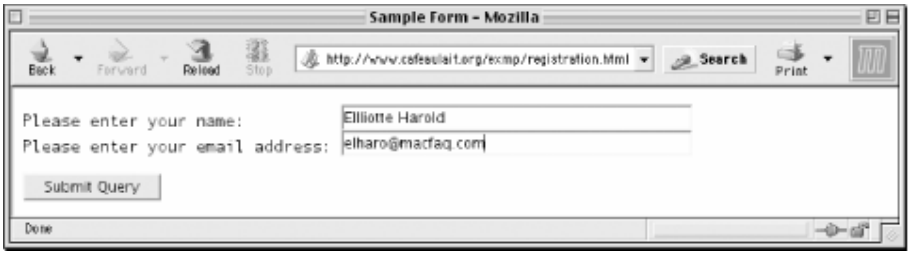
HTTP provides a standard, well understood and well supported means for Java applets and applications to talk to remote systems; therefore, I will cover how to use Java to both receive and send data to the server. There are other ways for Java programs to talk to servers, including Remote Method Invocation (RMI) and SOAP. However, RMI is slow and SOAP is quite complex. By way of contrast, HTTP is mature, robust, better supported across multiple platforms and web servers, and better understood in the web development community.

Example 2-1 and Figure 2-3 show a simple form with two fields that collects a name and an email address. The values the user enters in the form are sent back to the server when the user presses the "Submit Query" button. The program to run when the form data is received is /cgi/reg.pl; the program is specified in the `ACTION` attribute of the `FORM` element. The URL in this parameter is usually a relative URL, as it is in this example.

Example 2-1. A simple form with input fields for a name and an email address

```
<HTML>

<HEAD>

<TITLE>Sample Form</TITLE>

</HEAD>

<BODY>

<FORM METHOD=GET ACTION="/cgi/reg.pl">

<PRE>

Please enter your name: <INPUT NAME="username" SIZE=40>

Please enter your email address: <INPUT NAME="email" SIZE=40>

</PRE>

<INPUT TYPE="SUBMIT">

</FORM>

</BODY>

</HTML>
```

Figure 2-3. A simple form



The web browser reads the data the user types and encodes it in a simple fashion. The name of each field is separated from its value by the equals sign (=). Different fields are separated from each other by an ampersand (&). Each field name and value is x-www-form-url-encoded; that is, any non-ASCII or reserved characters are replaced by a percent sign followed by hexadecimal digits giving the value for that character in some character set. Spaces are a special case because they're so common. Instead of being encoded as %20, they become the + sign. The plus sign itself is encoded as %2b. For example, the data from the form in Figure 2-3 is encoded as:

```
username=Elliotte+Harold&email=elharo%40macfaq.com
```

This is called the query string.

There are two methods by which the query string can be sent to the server: `GET` and `POST`. If the form specifies the `GET` method, the browser attaches the query string to the URL it sends to the server. Forms that specify `POST` send the query string on an output stream. The form in Example 2-1 uses `GET` to communicate with the server, so it connects to the server and sends the following command:

```
GET
/cgi/reg.pl?username=Elliotte+Harold&email=elharo%40macfaq.com
HTTP/1.0
```

The server uses the path component of the URL to determine which program should handle this request. It passes the query string's set of name-value pairs to that program, which normally takes responsibility for replying to the client.

With the `POST` method, the web browser sends the usual headers and follows them with a blank line (two successive carriage return/linefeed pairs) and then sends the query string. If the form in Example 2-1 used `POST`, it would send this to the server:

```
POST /cgi-bin/register.pl HTTP 1.0

Content-type: application/x-www-form-urlencoded

Content-length: 65

username=Elliotte+Harold&email=elharo%40metalab.unc.edu
```

There are many different form tags in HTML that produce pop-up menus, radio buttons, and more. However, although these input widgets appear different to the user, the format of data they send to the server is the same. Each form element provides a name and an encoded string value.

Because `GET` requests include all necessary information in the URL, they can be bookmarked, linked to, spidered, googled, and so forth. The results of a `POST` request cannot. This is deliberate. `GET` is intended for noncommital actions, like browsing a static web page. `POST` is intended for actions that commit to something. For example, adding items to a shopping cart should be done with `GET`, because this action doesn't commit; you can still abandon the cart. However, placing the order should be done with `POST` because that action makes a commitment. This is why browsers ask you if you're sure when you go back to a page that uses `POST` (as shown in Figure 2-4). Reposting data may buy two copies of a book and charge your credit card twice.

Figure 2-4. Repost confirmation

In practice, `POST` is vastly overused on the web today. Any safe operation that does not commit the user to anything should use `GET` rather than `POST`. Only operations that commit the user should use `POST`.