Design and Implementation of FtpServlet package and a Container

Xueqing Han

A Major Report

in

The Department

of

Computer Science

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Computer Science at
Concordia University
Montreal, Quebec, Canada

September 2003

Your file Votre référence
ISBN: 0-612-83901-X
Our file Notre référence
ISBN: 0-612-83901-X

Abstract

Design and Implementation of FtpServlet package and a Container

Xueqing Han

Servlet technology is initially intended to provide the Web developers with a simple, consistent mechanism for extending the functionality of a Web server, which is built on top of Hypertext Transfer Protocol. This report extends the idea of servlet to another popular communication protocol in the Internet community, File Transfer Protocol. A FtpServlet API package, which is similar to the well-known HttpServlet package, is defined. A container that is able to accommodate FtpServlets is implemented to demonstrate the feasibility of FtpServlet package. Several samples of FtpServlets are provided to show how to use the package and its flexibility. Future improvement plans to consolidate the FtpServlet package are also discussed in order to make full use of the package.

Acknowledgements

I would like to express my thanks to my supervisor, Professor Jaroslav Opatrny for his guidance and invaluable comments throughout this work.

Many thanks also to all people with whom I exchanged opinions and ideas on related matters.

Dedication

To my mother and father

Table of Contents

List of Figures

List of Tables

Introduction

The servlet technology [1] was introduced by Sun MicroSystems Inc. for web servers. Servlets are the Java platform technology of choice for extending and enhancing Web servers. When we mention web, we usually refer to World Wide Web (WWW), which is built up on top of the Hypertext Transfer Protocol (HTTP)[2].

Basics about Web Server

A *Server* is usually a computer that can provide content services to client requests. Those services can be also extended to include computing request, querying requests, and so on. When a server is talked about, usually its counterpart, a client should also be introduced. The client of a web server, which can send requests for web content, is generally called web browser. There are several popular products in the market, i.e. Microsoft Internet Explorer (IE), Netscape Navigator, and Opera. The following figure shows the basic layout of client-server paradigm.

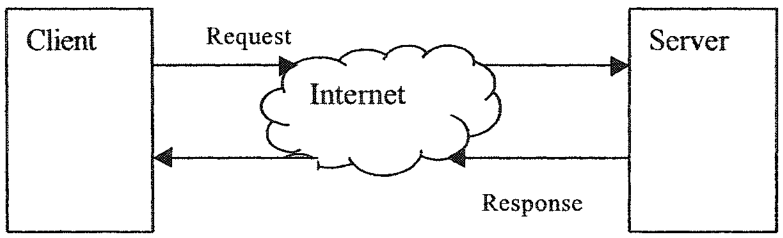

Figure 1 The client and server in the request-response paradigm

Web servers are to serve content over the Internet using the *Hypertext Markup Language* (HTML)[3]. The Web server accepts requests from browsers like Netscape and Internet Explorer and then returns the appropriate HTML documents. The developers of web servers usually organize the server content in the HTML format under local file system. When a client request comes, the web server locates the content in quest and sends it back to the client application. Along with the popularity of web servers and their applications, there is huge demand for web servers to delivery dynamic content based on the user request. In order to meet the demand, a web server has to be able to compute its response using some extensible components on the server-side. (Meanwhile client-side technologies, such as JavaScript[4], Visual Basic Script[5], were also introduced to face the challenge on the client side). A number of server-side technologies can be used to increase the power of the server beyond its ability to deliver standard HTML pages. These include Common Gateway Interface (CGI) scripts [6], server-side includes (SSI)[7], and Active Server Pages (ASPs)[8] from Microsoft.

Basics about HTTP

Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems, which is used by a WWW client (e.g. a web browser) to send a request to a web server. HTTP is a request-response-oriented protocol. An HTTP request is a packet consisting of a request method, a universal resource identifier (URI), header fields and a body (which can be empty). An HTTP response is a packet containing a result code and again header fields and a body.

When you request a Uniform Resource Locator (URL) in a web browser, the *GET method* is used for the request. A GET request does not have a body (i.e. the body is empty). The response should contain a body with the response data and header fields that describe the body (especially `Content-Type` and `Content-Encoding`). When you send a Hypertext Markup Language (HTML) form, either GET or POST can be used. With a GET request the parameters are encoded in the URL, with a POST request they are transmitted in the body. HTML editors and upload tools use PUT requests to upload resources to a web server and DELETE requests to delete resources.

Introduction of Servlet

Servlets are Java technology's answer to CGI programming. A servlet is a Java technology based web component, managed by a container, which generates dynamic content. A servlet can almost be thought of as an applet that runs on the server side -- without a face. Java Servlets are server-side programs that give Java-enabled servers additional functionality. Actually, servlet was first introduced as a key server-side Java solution, which has generated enthusiasm among developers because it allows sever-side Java programs to run with any major web server. Therefore, it provides a standard alternative to CGI programming that enhances performance and security and is easier to administer.

Unlike CGI scripts, servlets involve no platform-specific consideration or modifications. They are Java application components, which can be downloaded on demand to be part of the system that needs them.

The Java servlet API provides a simple framework for building applications on web servers. This API is described in the Java Servlet API Specification (currently version 2.4) by the Java Software Division of Sun Microsystems Inc. The servlet technology virtually consists of two parts. One is the servlet implementation; another is the servlet container (it is also called servlet engine).

Servlet Container

As we mentioned above, Servlets just are server-side components, which need to be running within a container, sometimes called servlet engine. Servlets interact with a *servlet engine* (an implementation of the Java Servlet API specification) in terms of requests and responses. The servlet engine in-turn interacts with the web server by delegating requests to servlets and transmitting responses to the web server. Usually, the servlet container can be implemented as a part of a web server or application server that provides the network services over which requests and responses are sent, decodes MIME based requests, and formats MIME based responses. Servlets interact with web clients via a request/response paradigm implemented by the servlet container.

Required by Java™ Servlet API specification, all servlet containers must support HTTP as a protocol for requests and responses, but additional request/response based protocols such as HTTPS (HTTP over SSL) may be supported.

Web Applications

Along with the popular success of servlet technology, the concept of *Web Application* [9] was introduced, which is strictly defined as a collection of web components. A web application is a collection of servlets, HTML pages, classes, and other resources that make up a complete application on a web server.

As we discussed above, the "Web Application" is confined within the HTTP community. Probably one reason for this trend is that the Internet Browsers simplify the development of the graphic user interface (GUI) part when the user wants the content shown in a friendly way. However, for many applications that do not need graphic user interface (GUI), the HTTP channel between the browser and a web server becomes a pure communicating means to carry on the requests and responses streams.

Features of Servlet Container

A server container provides a home for servlets. Any servlet has to be running within a container. It's the container's responsibility to decide when to instantiate a servlet, how to communicate with it, and when to terminate it.

Maintaining the Life Cycle of a Servlet

A servlet is managed through a well-defined life cycle that defines how it is loaded, instantiated and initialized, handles requests from clients, and how it is taken out of service. It is the responsibility of the servlet engine to maintain a servlet's life cycle. This life cycle is expressed in the API by the `init`, `service`, and `destroy` methods of the `javax.servlet.Servlet` interface that all servlets must, directly or indirectly through the `GenericServlet` or `HttpServlet` abstract classes, implement.

Loading and Instantiation

The servlet container is responsible for loading and instantiating a servlet. The instantiation and loading can occur when the engine is started or it can be delayed until the container determines that it needs the servlet to service a request.

After the servlet object is loaded and instantiated, the container must initialize the servlet before it can handle requests from clients. Initialization is provided so that a servlet can read any persistent configuration data, initialize costly resources, and perform any other one-time activities. The container initializes the servlet by calling the `init` method of the `Servlet` interface with a unique (per servlet definition) object implementing the `ServletConfig` interface. This configuration object allows the servlet to access name-value initialization parameters from the servlet container's configuration information. The configuration object also gives the servlet access to an object implementing the `ServletContext` interface that describes the runtime environment that the servlet is running within.

Wrapping up of the Client Requests

The request object is defined to encapsulate all information from the client request. In the HTTP protocol, this information is transmitted from the client to the server by the HTTP headers and the message body of the request. The servlet engine is responsible to collect, parse, and encode the client requests to construct a request object so that the servlet could retrieve these requests through servlet APIs.

Delivery of the Server Responses

A response object is defined to encapsulate all information to be returned from the server to the client. In the HTTP protocol, this information is transmitted from the server to the client either by HTTP headers or the message body of the request. The servlet engine is responsible for delivery all the response to the clients.

Java Servlet Framework

In the HTTP based request-response paradigm, a client user agent (a web browser or any such application that can make HTTP requests and receive HTTP responses) establishes a connection with a web server and sends a request to the server. If the web server has a mapping to a servlet for the specified URL in the request, the web server delegates the request to the specified servlet. The servlet in turn processes the request and generates an HTTP response.

Interaction between Servlet and Servlet Container

The following Figure 2 shows the relationship between servlets and the container, and how the client interacts with the server. Here is a description of a typical sequence of events among them.

A client program, such as a web browser, accesses a web server and makes an HTTP request.

This request is processed by the web server and is handed off to the servlet container.

The servlet container determines which servlet to invoke based on its internal configuration and calls it with objects representing the request and response. The servlet container can run in the same process as the host web server, in a different process on the same host, or on a different host from the web server for which it processes requests.

The servlet uses the request object to find out who the remote user is, and what HTML form parameters may have been sent as part of this request, and other relevant data. The servlet can then perform whatever logic it was programmed with and can generate data to send back to the client. It sends this data back to the client via the response object.

In many similar systems, the servlet will communicate with back-end database system. It usually involves retrieving information from the database system, saving information into it, and so on. This part is within the application domain of the servlet.

Once the servlet is done with the request, the servlet container ensures that the response is properly flushed and returns control back to the host web server.

The web server will return the response to the client so that the user can get the result of the request.

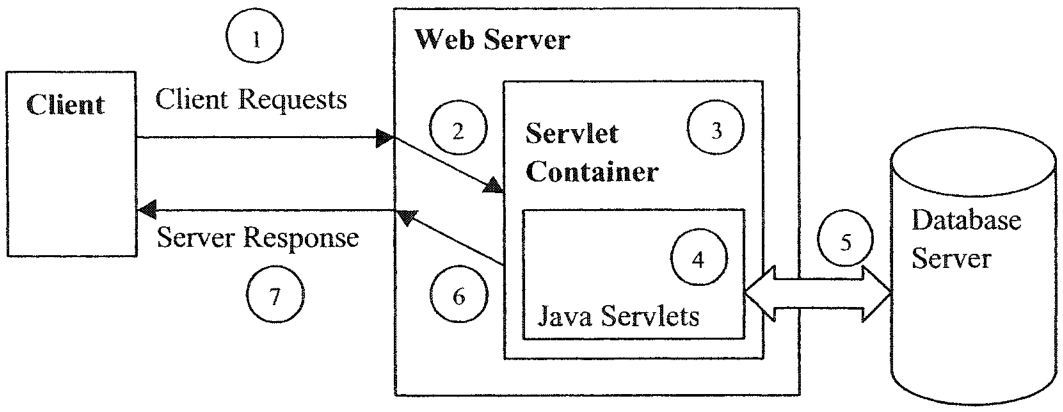

Figure 2 The interaction among servlet container, servlet, and web server

Construction of Servlet Framework

In order to fulfill the requirement of web application development, the servlet API provides three primary abstractions: HTTP requests, request processing based on some application logic, and HTTP responses. These abstractions simplify the application development as far as the requests and responses are concerned. In addition, the servlet API also provides a mechanism for session tracking and state management. These are described below.

HTTP Request

The interface `HttpServletRequest` is the first abstraction provided by the servlet API. This interface encapsulates HTTP requests from a user agent. When the servlet engine receives a request, an object implementing this interface is constructed and passed on to a servlet. This object provides methods for accessing parameter names and values of the request, other attributes of the request, and an input stream containing the body of the request.

HTTP Response

The `HttpServletResponse` interface of the servlet API provides an encapsulation of the HTTP response generated by a servlet. This interface defines an object created by the servlet engine that lets a servlet generate content in response to a user agent's request. This object provides methods to set type and length of the content, character encoding, cookies, response status including errors, and an output stream into which binary response data may be written. Alternatively, this object also provides a print writer object for writing formatted text responses.

Application Logic and Content Generation

The `HttpServlet` interface specifies methods for implementing the application logic and generating content in response to an HTTP request. These methods handle the GET, POST, HEAD, DELETE, OPTIONS, PUT and TRACE requests of HTTP. These methods are invoked by the servlet engine and act as placeholders for implementing application logic. The servlet engine also supplies `HttpServletRequest` and `HttpServletResponse` objects to these methods.

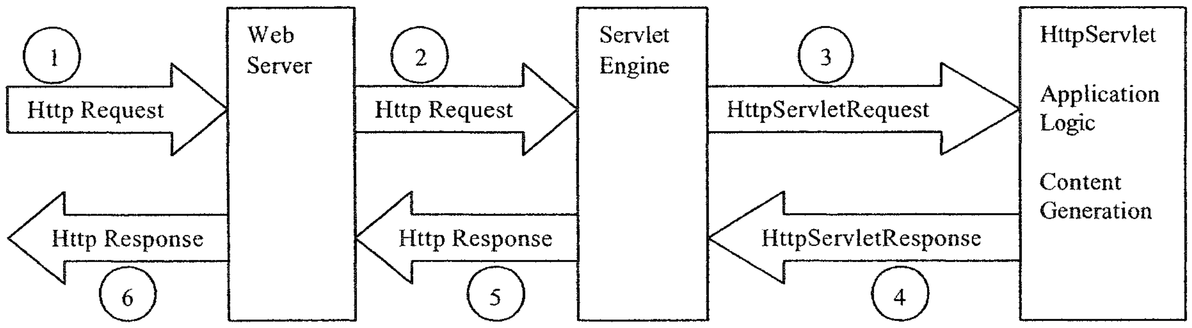

Figure 3 The request-response sequence among web server, servlet engine, and servlet

As depicted in Figure 3, the request first coming from the user client is in the format of HTTP request, which is defined by HTTP. The web server will dispatch the request to its handler, either a static resource, or a servlet based on the incoming URL. If the request is for a servlet, the web server will redirect it to the servlet engine. This is shown in Step 2. The request should be maintained by the web server and passed to the servlet engine. Once the servlet engine got the request, it will decide which servlet is targeted and load that servlet and its resources. After the servlet is instantiated, the servlet engine will wrap up the request to form a `HttpServletRequest` object and pass it to the servlet. After the servlet finishes its logic and comes up with the response, it will send back the response in an object, `HttpServletResponse`. Actually, this object is also provided by the servlet engine so that the servlet can easily communicate with the system. Once the servlet obtains the object `HttpServletResponse`, it will pass it back to the web server in the format of HTTP response. In turn, the web server will return the response to the client, which could be a web browser or other client application. The response is then shown by the client application to the user as the result of the initial request.

These interfaces are shown in the above diagram in the context of a request and a response. In this diagram, the service method of the `HttpServlet` is shown to implement the application logic and content generation, although one of the `doGet`, `doPost`, `doHead`, `doDelete`, `doOptions`, `doPut` or `doTrace` methods can handle HTTP requests.

Session Tracking and State Management

HTTP, which is the underlying protocol for web applications, is stateless. This protocol covers only a single request (with the connection initiated by the user agent) and a response. In this protocol, irrespective of the status of the protocol, the connection may be closed by either of the transacting parties. This has the following ramifications:

The protocol has no mechanism by which a series of unique requests from a user agent may be identified as being from the same user agent. Consequently, in a transaction spanning multiple requests and responses, the web server can not uniquely determine that all the requests are from the same user agent. A user, therefore, can not establish a dialog with the web server to perform a business transaction.

Since connections are not maintained by either of the transacting parties, the state of the transaction (and the application) can not be preserved across multiple requests.

The Java servlet API provides the `HttpSession` interface for session tracking and state management in a session. A servlet obtains a `HttpSession` object from the associated `HttpServletRequest` object. Servlets track sessions by URL rewriting or by cookies. Objects of type `HttpSession` can be set or obtained for new and existing sessions respectively from `HttpServletRequest` objects. Session specific data can be stored into these objects.

Java Servlet API

Sun Microsystems, Inc. has released several versions of the servlet specification and its implementation samples [1]. In this section we will explore the detailed structure of a servlet and its related design issues.

Generally, two packages, `javax.servlet` and `javax.servlet.http` are provided. The `javax.servlet` package contains a number of classes and interfaces that describe and define the contracts between a servlet class and the runtime environment provided for an instance of such a class by a conforming servlet container. The `javax.servlet.http` package contains a number of classes and interfaces that describe and define the contracts between a servlet class running under the HTTP protocol and the runtime environment provided for an instance of such a class by a conforming servlet container.

Class Hierarchy

Here is the class hierarchy defined in servlet package

Class Hierarchy

- class java.lang.Object
    - class javax.servlet.http.Cookie (implements java.lang.Cloneable)
    - class java.util.EventObject (implements java.io.Serializable)
        - class javax.servlet.http.HttpSessionBindingEvent

- class javax.servlet.GenericServlet (implements javax.servlet.Servlet, javax.servlet.ServletConfig, java.io.Serializable)
- class javax.servlet.http.HttpServlet (implements java.io.Serializable)
- interface javax.servlet.http.HttpServletRequest (extends javax.servlet.ServletRequest)
- interface javax.servlet.http.HttpServletResponse (extends javax.servlet.ServletResponse)
- interface javax.servlet.http.HttpSession
- interface javax.servlet.http.HttpSessionBindingListener (extends java.util.EventListener)
- interface javax.servlet.http.HttpSessionContext
- class javax.servlet.http.HttpUtils
- class java.io.InputStream
    - class javax.servlet.ServletInputStream
- class java.io.OutputStream
    - class javax.servlet.ServletOutputStream
- interface javax.servlet.Servlet
- interface javax.servlet.ServletConfig
- interface javax.servlet.ServletContext
- interface javax.servlet.ServletRequest
- interface javax.servlet.ServletResponse
- interface javax.servlet.SingleThreadModel
- class java.lang.Throwable (implements java.io.Serializable)

- class java.lang.Exception
    - class javax.servlet.ServletException
        - class javax.servlet.UnavailableException

Servlet interface

The `Servlet` interface is the central abstraction of the Servlet API. All servlets implement this interface either directly, or more commonly, by extending a class that implements the interface. The two classes in the API that implement the Servlet interface are `GenericServlet` and `HttpServlet`.

Actually, the interface defines several methods for the servlet container to make the management of a servlet's life cycle easy and consistent. They are known as life-cycle methods, and are called in the following sequence:

The servlet is constructed, then initialized with the `init()` method.
Any calls from clients to the `service()` method are handled.
The servlet is taken out of service, then destroyed with the `destroy()` method, then garbage collected and finalized.

In addition to the life-cycle methods, this interface provides the `getServletConfig` method, which the servlet can use to get any startup information, and the `getServletInfo` method, which allows the servlet to return basic information about itself, such as author, version, and copyright.

GenericServlet

`GenericServlet` is a generic, protocol-independent servlet. It implements the `Servlet` and `ServletConfig` interfaces. `GenericServlet` may be directly extended by a servlet, although it is more common to extend a protocol-specific subclass such as `HttpServlet`. `GenericServlet` makes writing servlets easier. It provides simple versions of the lifecycle methods `init` and `destroy` and of the methods in the `ServletConfig` interface. `GenericServlet` also implements the `log` method, declared in the `ServletContext` interface. To write a generic servlet, you need only override the abstract `service` method.

HttpServlet

The class `HttpServlet` implements the `Servlet` interface and provides a base that developers will extend to implement servlets for implementing web applications employing the HTTP protocol. In addition to generic Servlet interface methods, the class `HttpServlet` implements interfaces providing HTTP functionality.

HttpServlet provides an abstract class to be subclassed to create an HTTP servlet suitable for a Web site. A subclass of `HttpServlet` must override at least one method, usually one of these:

`doGet`, if the servlet supports HTTP GET requests

`doPost`, for HTTP POST requests

`doPut`, for HTTP PUT requests

`doDelete`, for HTTP DELETE requests

`init` and `destroy`, to manage resources that are held for the life of the servlet

`getServletInfo`, which the servlet uses to provide information about itself

There's almost no reason to override the `service` method. `Service` handles standard HTTP requests by dispatching them to the handler methods for each HTTP request type (the `do`*XXX* methods listed above).

For most purposes, developers will typically extend `HttpServlet` to implement their servlets.

Jetty, an example of the Servlet container

Jetty is an Open Source HTTP/Servlet/JSP engine. The full version of Jetty is available from http://www.mortbay.com, which includes downloads of the full source package.

General components layout

The Jetty servlet engine consists of following main components:

- The `org.mortbay.http.HttpServer` class implements the core HTTP server.
- The `org.mortbay.jetty.Server` class extends `org.mortbay.http.HttpServer` with XML configuration and servlet support.
- `SocketListner` is the basic implementation of `HttpListner` that accepts HTTP connections on a normal socket.
- `HttpListner` implementations create `HttpConnection` instances.
- A `HttpConnection` instance manages one or more requests received over a connection.
- A `HttpContext` has attributes for class path, class loader and resource base, which apply to all the `HttpHandlers` it contains.
- A `HttpContext` is registered in the `HttpServer` at one and only one context pattern (eg. /context/*) and for zero or more virtual hosts.
- A `HttpContext` contains one or more implementation of `HttpHandler`, in the order they are added.

- `ResoueceHandler` is an implementation of `HttpHandler` that serves static content from the resource base of the `HttpContext`.
- `ServletHandler` is an implementation of `HttpHandler` that serves Servlets by mapping a path spec within the context (eg *.xxx, / or /XX/*) to a Servlet instance. Dynamic servlets include the class name in the URL, configured servlets have explicit path to class mappings.

Jetty Architecture

The class diagram in Figure 4 shows the relationship among the key Jetty classes.

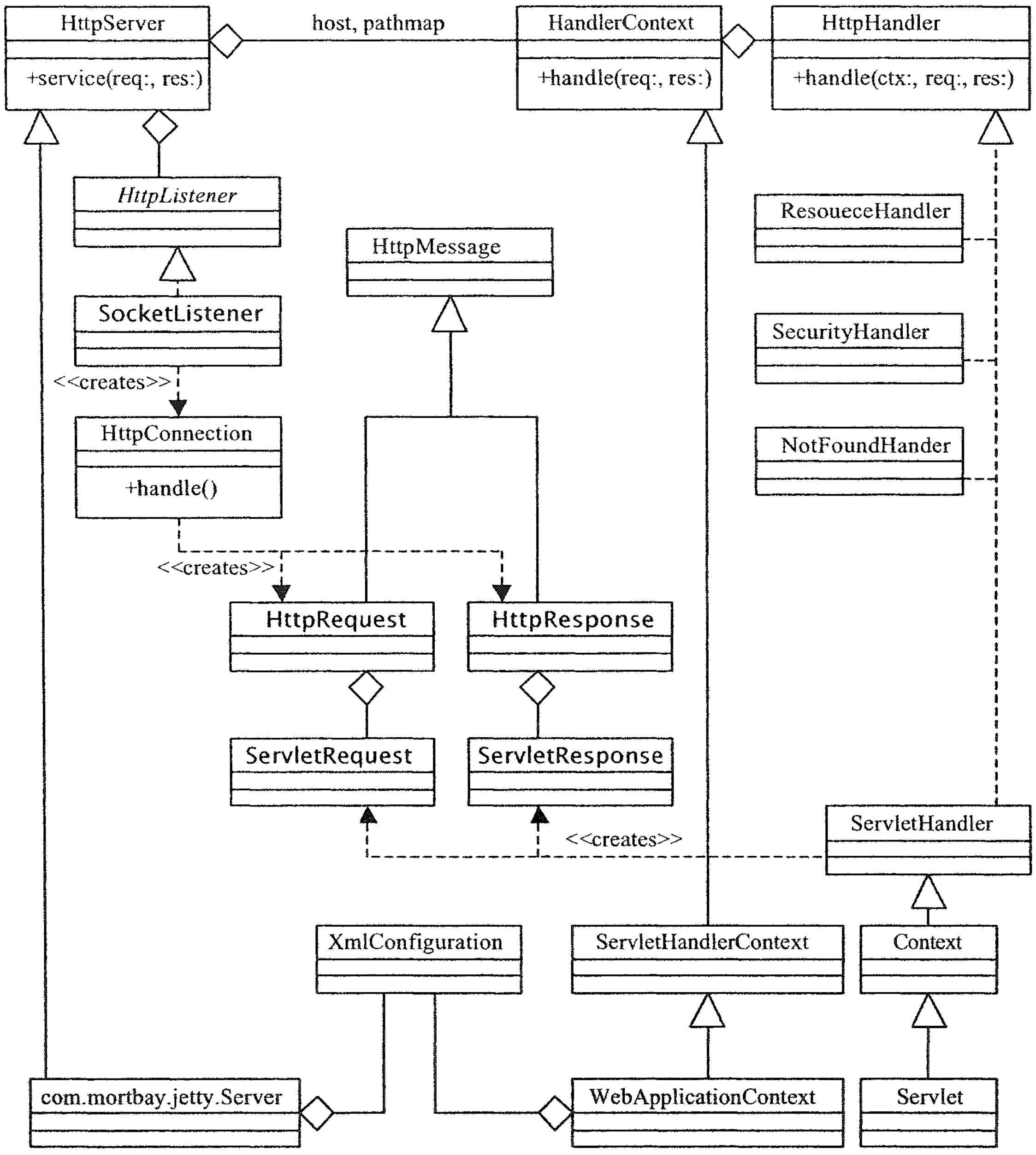

Figure 4 Class Diagram for Jetty servlet engine

Extending the Servlet Concept to FTP

From downloading the newest software to transferring corporate documents, a significant percentage of Internet traffic consists of file transfers. Files can be transferred via HTTP, but the page orientation of the Web protocol has disadvantages, especially for performance and control. That's where one of the oldest of the Internet services steps in: File Transfer Protocol (FTP)[10]. Essentially, FTP makes it possible to move one or more files between computers with security and data integrity controls appropriate for the Internet.

Due to the flexibility of the servlet package, servlet technology has been widely used within HTTP server and its extensions. The usage of the technology has provided the developer of HTTP servers the subtle and complete control on what content could be provided to the clients. Since the servlet API package assumes nothing about the server's environment or protocol, the structure of servlets could also be embedded in many other servers, which would not rely on HTTP. That gives us the possibility to extend the servlet technology to another very popular used protocol, FTP, the File Transfer Protocol.

Basics about FTP

FTP is one of the oldest and still most popular protocols on the Internet. While some protocols, like Gopher has lost general interest, FTP is used more than ever before.

Hundreds of thousands of FTP servers are running, and millions of people use FTP every day to send or receive files.

FTP is simply a service that allows two Internet hosts to transfer files back and forth. In this way, it is really a large part of the structure of how the Internet works.

FTP works in a client-server fashion where the local host (either your computer or a service provider's computer) initiates the client program, FTP, and connects to the remote system, in this case representing the server. The user initiates commands with the client application and the server responds.

To utilize FTP, in addition to having access to your computer, you need to know three things:

- The name of the machine that the file is kept on, and
- Where the file is on that machine, and
- What the file is called.

Typically a user enters commands to FTP to browse the remote local file system to retrieve files from the remote system.

Anonymous FTP

When copying files from one computer to another, one must have the permission to access another computer system, typically that means having a valid account and password. Anonymous FTP is a special use of FTP in which a site makes its files available to users who do not have accounts on that host. The anonymous FTP facility lets you use a special login name of anonymous and a special password, usually your email address, to let you have limited access to the remote host for the purpose of retrieving files that have been approved for public access. Usually with an Anonymous FTP site, there will be certain directories that are designated as public directories. Most sites will also allow you to download files from it, but not copy any files up to the anonymous host site.

Using FTP

To start an FTP session, One must establish a network connection from your local host to the remote host. On many Internet hosts, One types FTP at the command prompt and the host name of the machine to which you want to connect. Many of the Internet software packages one will be using under Microsoft Windows will have a GUI, graphical user interface that will ask for these and have GUI-based FTP tools that allow for easy browsing of directories and hosts. Macintosh users begin their sessions by clicking on the Fetch icon. When you are at the remote host, it will prompt you for your username and a password. Again, when using anonymous FTP, you enter anonymous as your user name and your email address as the password.

Request and response paradigm

Similar to HTTP, FTP is also a protocol based on request and response logic. FTP requires two programs, a server program, and a client program. Normally the server program offers files to the client program. But in some cases, the server will also allow the client to upload files. The client program connects to an FTP sever on the Internet. Once connected, the FTP server sends a welcome message to the client over the open socket (network) connection.

Connections in FTP

Unlike HTTP where all the connections are the same, FTP connections are categorized as two types, i.e. *control connection* and *data connection*. Actually, *data connection* is only temporary. The creation and termination of a data connection are controlled by the control connection. *Control connection* is used for the client to send commands, and for the server to response in plain text, which is a bi-directional communication channel. The *data connection* is just for data transfer, with one connection for each data transfer.

Design of FtpServlet Container

As we discussed above, the servlet container is playing an important role in the servlet technology. It must receive the client request, and construct the context object and request object for the servlet to retrieve all the request information. Meanwhile, the servlet engine also needs to pass back any response to the client.

When extending the servlet technology to FTP applications, it is an intuitive approach to implement a container that can accommodate this specific servlets. Similar to the HttpServlet package, the definition of FtpServlet package would provide the interface between a Servlet instance and the servlet container. In order to work out a feasible definition for the FtpServlet package, the possible implementation for a servlet container will be examined in this section.

A container to support FTP connection

The ultimate purpose would be to implement an FTP server to handle all the requests from the clients. Once we introduce the idea of FTP Servlet container, we would have an equation shown in Figure 5. We would like to allocate the assignment among those components left to the equation.

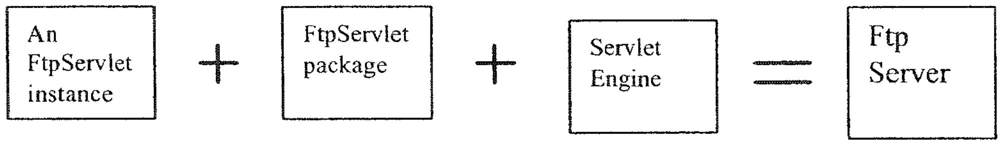

Figure 5 The relation between ftp server and its components

As shown in Figure 5, the application logic is implemented by an FtpServlet instance. It will decide how to response to the request from the user based on the nature of the application. The servlet engine would provide an environment for the FtpServlet instance to fulfill its actions. More specifically, the servlet engine would be responsible for connecting to the Ftp Server, establishing the link between the client and the server, and tearing down the link. When it comes to the communication protocol between the FtpServlet and the servlet engine, it is the purpose of this FtpServlet package, which will define the necessary interface to guide how the FtpServlet instance communicates with the servlet container.

All the FTP details should be taken care of by the FtpServlet package and the servlet engine. As the user of FtpServlet, who has also to be the developer of the FTP server, the work is just to implement an FtpServlet instance, just as simple as finishing a HttppServlet.

General consideration of the FtpServlet Container

Based on the implementation of Jetty servlet engine, the Servlet Container would be implemented in the following aspects:

SocketListener would play a much important role. As we know, Ftp request is coming though the port 21. The Listener would listen to this port and once a request is coming, an FtpConnection would be created, which is then to maintain the ftp connection.

Following the same logic, the FtpConnection would rely on both request and response objects. Without exception, they would be named as FtpRequest and FtpResponse. However, the two objects are not part of the FtpServlet package. They belong to the container implementation. The two objects reflect the container's knowledge about the FTP process.

The Ftp protocol has two kinds of connections, which are *control connection* and data *connection*. Therefore, the FtpConnection would have two descendents, FtpCtrlConnection and FtpDataConnection, to accommodate the complexity.

Definitely, an FtpServer class is needed to manage all the controlling features. Actually, the type of servers, either HttpServer or FtpServer, may be decided by the port number, through which the client request is coming. For Ftp connection, the port number is predefined and well known. For the HttpServer in the Jetty package, the port number is configurable. We may define a port usage table to pre-assign these resources. For a request coming from the port number within the range allocated for Ftp Server, FtpServer will be initialized and used to handle the services. If a request is knocking on the ports defined for Http services, the HttpServer will be created and the classical application server is instantiated.

Similarly, the context object is needed, which is responsible for maintaining the communication between the servlet and its environment, i.e. the servlet container and the operating system. Some system information should be resolved by querying the context object.

Wrapping up the Ftp request and response

The servlet container needs to read in the request coming from the FTP client and process it to create an FtpServletRequest object to contain all the request intention from the original client. The following diagram shows the relationship between these components.

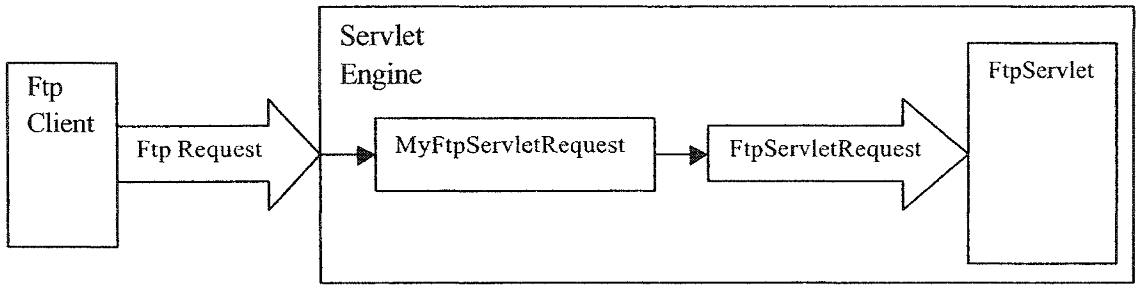

Figure 6 The Ftp Request is wrapped up by the servlet engine

However, in the implementation of Servlet engine, it could not just convert Ftp request to FtpServletRequest, because it will be defined as an interface in the FtpServlet package. A "real" object that implements the FtpServletRequest interface must be created by the servlet container to wrap up all the request information.

The Figure 7 shows how the MyFtpServletRequest is to be constructed.

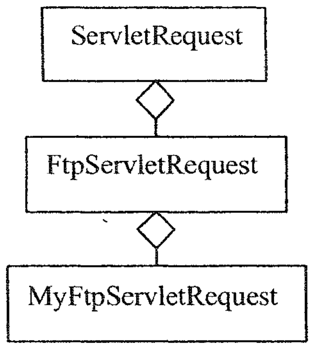

Figure 7 The hierarchy for the construction of request object

Where FtpServletRequest belongs to the FtpServlet package we are going to define later, while ServletRequest is defined by the Sun Servlet API.

Following the same deduction for FTP request, the FTP response is to be wrapped up by the container using an object, MyFtpServletResponse.

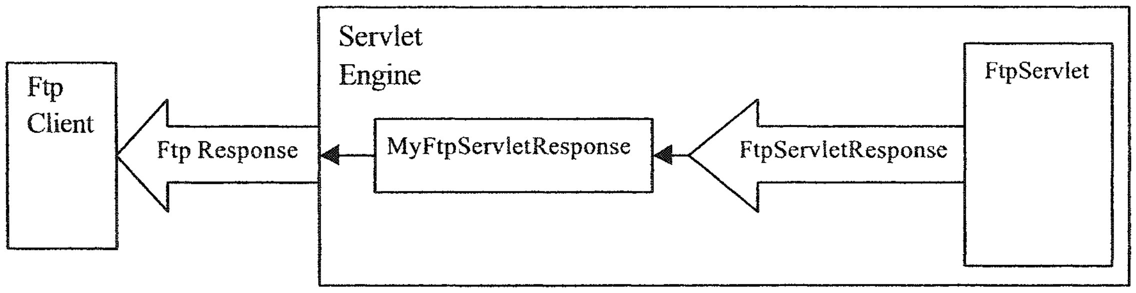

Figure 8 The Ftp Response is wrapped up by the servlet engine

The response defined by the application's nature will be written back to the *Servlet Engine* through the interface, FtpServletResponse, which in turn is implemented by the engine as an object, MyFtpServletResponse. The Servlet Engine is taking care of sending back the response to the FTP client, where it is merely finishing a conversion from the internal representation of the response to its formal counterpart, i.e. Ftp Response.

Class loader

Some of Java features really stand out to make Java a good choice to implement system software, such as dynamic loading, Network-centric Programming, and security.

Java is both dynamic and extensible. Java code is organized into modular object-oriented units called classes. Classes are stored in separate files and are loaded into the Java interpreter only when needed. This means that an application can decide, as it is running, what classes it needs, and can load them when it needs them. In also means that a program can dynamically extend itself by loading the classes it needs to expand its functionality.

A *classloader* is a Java class that locates and loads a requested class into the Java virtual machine (JVM). A classloader resolves references by searching for files in the directories or JAR files listed in its classpath. Most Java programs have a single classloader, the default system classloader created when the JVM starts up. Classloaders are hierarchical. A classloader always asks its parent for a class before it searches its own classpath. But a parent classloader does not consult its children. The implementation of a servlet engine also relies on the mechanism to dynamically find the target servlet and initiate it.

Event sequence of FtpServlet Engine

Figure 9 shows the message sequence among several engine objects.

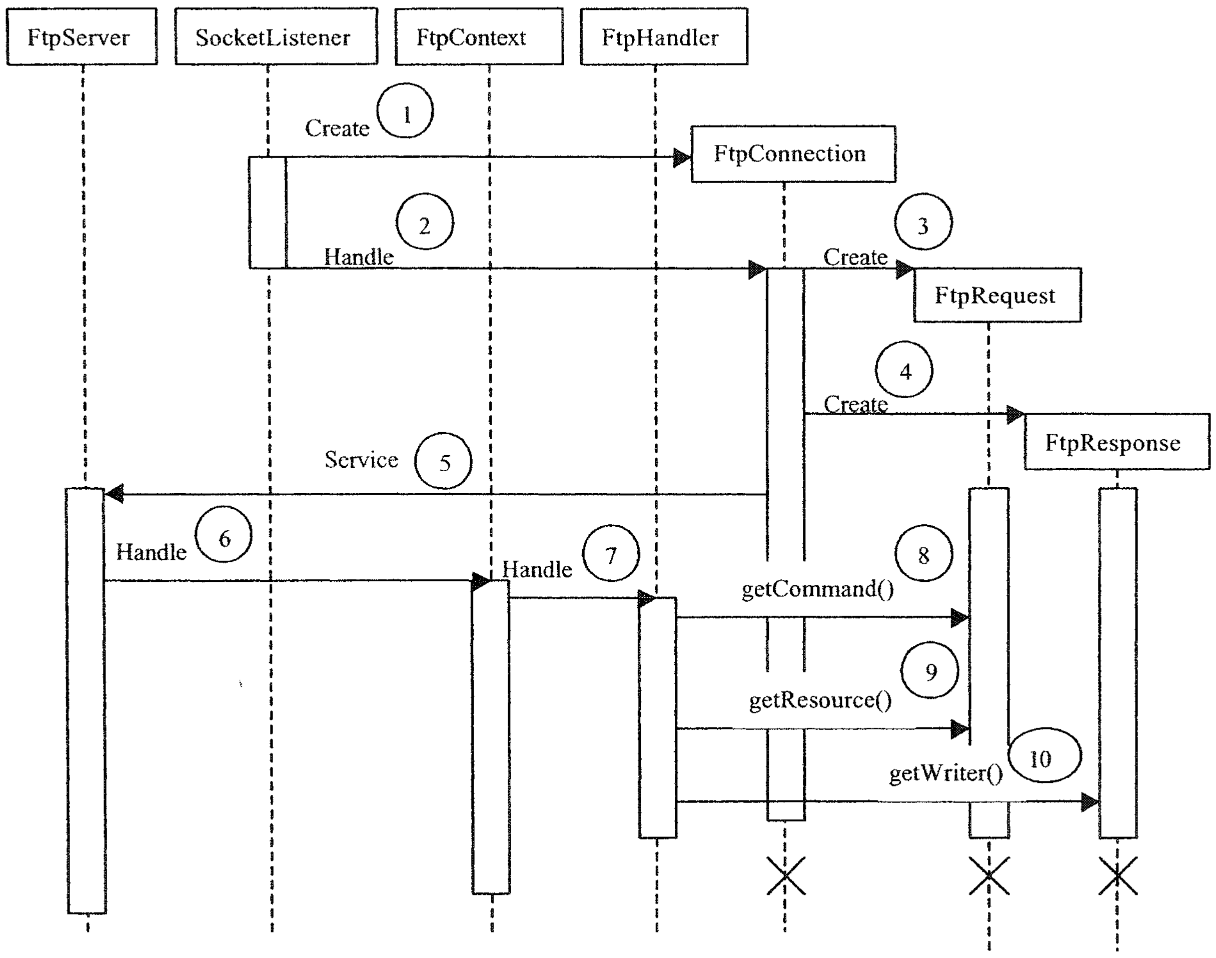

Figure 9 The sequence diagram of the interaction among objects of ftp serverlets

As shown in the above diagram, the SocketListener first detects there is a connecting request from a client, it will then create a connection object, FtpConnection, to handle the request. As shown in the step 2, the handle method of FtpConnection is called by

SocketListener, where the FtpRequest and FtpResponce objects will be created during Step 3 and 4. When the request and response objects are ready, the FtpConnection object in Step 5 will call the service method of FtpServer to ask the request to be served. The FtpServer object maintains all the context information for the server, it then passes the request to FtpContext to resolve which servlet should answer the request. After the context object figures out which handler should be invoked, it calls the hander method of FtpHandler in Step 7 to take over the responsibility. The FtpHandler object then uses the FtpRequest object to decide what the client request is and what resource it has provided through calling getCommand() and getReouce() in Step 8 and 9. All the information will be passed to the FtpServlet and used to generate proper response, which in turn will be returned to the client through FtpResponce object by calling getWriter() in Step 10.

Implementation of FtpServlet

Actually, FTP is a smaller protocol comparing with HTTP. In contrast to that HTTP is using Hypertext Markup Language (HTML) as underlying format, the transmission format that FTP relies on is much simpler. Because of the simplicity of FTP format, there is not much room for applications to extend from it.

Reference to Sun's servlet package

Sun Microsystems, Inc. has released several versions of the servlet Specification. The servlet specification defines two packages, i.e. javax.servlet and javax.servlet.http. Table 1 gives the summary of servlet API package.

Table A1 Java servlet API summary

Package javax.servlet	**Package javax.serlvet.http**
`RequestDispatcher`	`HttpServletRequest`
`Servlet`	`HttpServletResponse`
`ServletConfig`	`HttpSession`
`ServletContext`	`HttpSessionBindingListener`
`ServletRequest`	`HttpSessionContext`
`ServletResponse`	`Cookie`
`SingleThreadModel`	`HttpServlet`
`GenericServlet`	`HttpSessionBindingEvent`
`ServletInputStream`	`HttpUtils`
`ServletOutputStream`	

ServletException	
UnavailableException	

From last section where the implementation of Generic Servlet Engine was discussed, we know that the definition of FtpServlet will hugely affect the implementation. Since the HttpServlet package has been proved to be a successful example, the design and implementation of FtpServlet could borrow some ideas from the implementation of HttpServlet. Here some preliminary ideas will be discussed.

FtpServletRequest

Besides all the methods defined in SevletRequest, which should be implemented in this interface, another two important methods are defined in it.

```
public String getCommand();

public String getResource();
```

getCommand() method will return the command the client has requested. The command is in the format of string and in its upper case (Please refer to the appendix for all the FTP commands.).

getResource() methods returns the parameter(s) for a command if applicable. For example, if the user issues a USER command, the user name will be the command resource. When a PASS command is issued, the parameter will be retrieved using getResource() (i,e, joe@nowhere.com in the following sample).

```
Server: 220 Sample FTP server ready. Please give user-name
Client: USER anonymous
Server: 331 User name OK. Please give your email address as
password
Client: PASS joe@nowhere.com
Server: 230 User logged in
```

FtpServletResponse

As we discussed it before, there is a *control connection* and a *data connection* in FTP services. Since the *data connection* is temporary, it is not suitable to be provided in the response object. However, all the control data flow is transmitted through this object. Another point worth to mention is that the control data flow is actually in the format of ACSII text. All the responses (called replies in the FTP RFC [10]) are sent back using the text channel.

```
String command = req.getCommand();

String reply = "";

try {

        if(command.equals(METHOD_USER)) {

                m_user = req.getResource();

                reply = handleUSER( m_user);


        } else if(command.equals(METHOD_PASS) ) {

                reply = handlePASS(req.getResource());


        ......

        }
} catch( NoSuchMethodException e ) {
}
// send back the response
resp.getWriter().println(reply);
```

As the above sample code shows, the handling of each command will return a string response (the reply in the sample). The FtpServletResponse implements the getWriter() method from ServletResponse, which is used to get a print writer for sending back the text response.

Data Connection

A *data connection* is needed for some FTP requests, such as LIST, RETR, and STOR. Actually, these requests are always paired with a PORT command, which is used to ask the server to prepare a data connection. The resources of a PORT command contains the host IP address and port number that the client is asking the server to listen to. When the second command comes, the server will use the host and port info to make a temporary data connection.

```
protected InetAddress    m_dataHost;   // host for the data connection
protected int            m_dataPort;   // port number for the data connection

private String handlePORT( String address ) throws UnknownHostException
{
        int i1 = address.lastIndexOf( ',' );
        if( i1 < 0 ) return "500 'PORT " + address + "': command not understood.";
        int i2 = address.lastIndexOf( ',', i1-1 );

        if( i2 < 0 ) return "500 'PORT " + address + "': command not understood.";

        try {
                m_dataHost = InetAddress.getByName( address.substring( 0, i2 ).replace( ',',
'.' ) );
                m_dataPort = Integer.parseInt( address.substring( i2+1, i1 ) ) * 256 +
                Integer.parseInt( address.substring( i1+1 ) );

        } catch( NumberFormatException e ) {
                return "500 'PORT " + address + "': command not understood.";
        }
        return "200 PORT command successful.";
}
```

The above code sample is used to handle the PORT request. Actually, it just parses the PORT resource and saves into local variables for later user.

```
        private Socket buildDataConnection( String name, String dataType, long bytes )
                throws Exception
        {
                Socket sock;
                if( passive != null )
                {
                        sock = passive.getConnection();
                        passive = null;
                }
                else if( m_dataHost == null )
                {
                        throw new Exception(
                                "Cannot build data connection: use PORT or PASV first." );
                }
                else
                {
                        sock = new Socket( m_dataHost, m_dataPort );
                        m_dataHost = null;
                }
                return sock;
        }
```

The above sample code is used to create data connection. Once the data connection finishes the data transmission, it is closed by the control connection.

FtpServlet

Since FtpServlet is derived from GenericServlet (that is the whole idea of this project), which defines a generic, protocol-independent servlet, it could make use of the well-designed life cycle control for servlet. In order to implement a protocol-specified servlet, the abstract `service` method of GenericServlet should be extended.

```
protected void service(FtpServletRequest req, FtpServletResponse resp)
        throws ServletException, IOException
{
        String command = req.getCommand();
        String reply = "";
        try {
                try {
                        if(command.equals(METHOD_USER)) {
                                m_user = req.getResource();
                                reply = handleUSER( m_user);

                        } else if(command.equals(METHOD_PASS) ) {
                                reply = handlePASS(req.getResource());

                        } else if(command.equals(METHOD_PORT) ) {
                                reply = handlePORT(req.getResource());

                        } else if(command.equals(METHOD_LIST) ) {
                                reply = handleLIST(resp, req.getResource());

                        } else if(command.equals(METHOD_CWD) ) {
                                reply = handleCWD( req.getResource());


                        } else if(command.equals(METHOD_CDUP) ) {
                                reply = handleCDUP( req.getResource());


                        } else if(command.equals(METHOD_PWD) ) {
                                reply = handlePWD( req.getResource());


                        } else if(command.equals(METHOD_XPWD) ) {
```

```
        reply = handleXPWD(req.getResource());


} else if(command.equals(METHOD_TYPE) ) {
        reply = handleTYPE(req.getResource());


} else if(command.equals(METHOD_PASV) ) {
        reply = handlePASV(req.getResource());


} else if(command.equals(METHOD_RETR) ) {
        reply = handleRETR(resp, req.getResource());


} else if(command.equals(METHOD_REST) ) {
        reply = handleREST(req.getResource());


} else if(command.equals(METHOD_STOR) ) {
        reply = handleSTOR(resp, req.getResource());


} else if(command.equals(METHOD_APPE) ) {
        reply = handleAPPE(resp, req.getResource());

} else if(command.equals(METHOD_SIZE) ) {
        reply = handleSIZE(req.getResource());


} else if(command.equals(METHOD_SYST) ) {
        reply = handleSYST(req.getResource());


} else if(command.equals(METHOD_RNFR) ) {
        reply = handleRNFR(req.getResource());

} else if(command.equals(METHOD_RNTO) ) {
        reply = handleRNTO(req.getResource());


} else if(command.equals(METHOD_MKD) ) {
        reply = handleMKD(req.getResource());

} else if(command.equals(METHOD_XMKD) ) {
        reply = handleXMKD(req.getResource());
```

```
                } else if(command.equals(METHOD_DELE) ) {
                        reply = handleDELE(req.getResource());


                        } else if(command.equals(METHOD_RMD) ) {
                        reply = handleRMD( req.getResource());

                } else if(command.equals(METHOD_NOOP) ) {
                        reply = handleNOOP(req.getResource());


                } else if(command.equals(METHOD_QUIT) ) {
                        reply = handleQUIT(req.getResource());
                } else {
                    reply = "502 Command '" + command + "' not
implemented.";
                    throw new NoSuchMethodException();
                }
            } catch( NoSuchMethodException e ) {
            }
            // send back the response
            resp.getWriter().println(reply);
        } catch( Exception e ) {
            try
            {
                Throwable p = e;
                while( p instanceof InvocationTargetException )
                p = ((InvocationTargetException)p).getTargetException();
                p.printStackTrace();
                reply = "555 Error: " + p.toString().replace( '\n', ' ' ) ;
            } catch( Exception e1 ) {
                e1.printStackTrace();
            }
        }
}
```

This method just parses the client's command and redirects it to its corresponding handler. Since this method can be overridden, the developer can implement almost every requirement needed.

Samples of FtpServet

In order to verify the idea of FtpServlet and its container, several sample servlets are implemented.

Account Checking

As in normal scenarios of FTP services, user permission is a very important process so that no security risks are introduced. With the usage of FtpServlet, the developer of an FTP server can easily add a new layer to achieve security checking. Here is an extraction from the FtpServlet sample.

```
protected boolean doPass(String user, String pass)
{
        if(user.equalsIgnoreCase("johndoe")) return false;
        if(pass.equalsIgnoreCase("guest")) return false;

        return true;
}
```

The servlet would just reject the user 'johndoe" and anyone who tries to login to the system using "guest" as password. Actually, in real life, the user/password information may be maintained in a secured place and retrieved on the fly. The roles of uses may also be stored to check if the user is allowed to access some sensitive information.

Filtering the response

Sometimes the Ftp Server may add more control over which kind of content should be sent back to the client. Two types of filters are provided in the FtpServlet package. One is just to check if the listed file name contains the required string. Another is to employ regular expressions for the filtering. Here are two samples.

```
public class NameAlikeFilterFtp extends FtpServlet

{

        protected String setListRegExpPattern()

        {

                return ".cpp";

        }

}
```

This FtpServlet would just list all the files that contain string ".cpp".

```
public class NameRegExpFilter extends FtpServlet

{

        protected String setListRegExpPattern()

        {

                return "[abc]";

        }

}
```

This one will just list all the files that contain any character, “a”, “b”, or “c”.

Database connection

In the nature of FTP service, the client actually does not know if the server really has that folder or file. The fact would give us the chance to let the FTP server provide the client whatever information the server believes proper. Actually, the `FtpServlet` can retrieve information from a predefined database system and convert it into the format that client could accept. Thus, we can add another application layer to extend the `FtpServlet` to provide proper information to the clients.

Further Improvement

Although the basic idea of FtpServlet was shown in different perspectives in this project, there is still some room to refine the interface definition so that more flexibility can be delivered to the developer of FTP server.

Refine the FtpServlet package

As we mentioned above, the developer can directly override the service() method to extend the FTP definition. Actually, all the handleXXX() functions can also be exposed to the developer so that the FTP service can be well adjusted according to the nature of the application.

As requested by the Java™ Servlet API specification, servlet is running above the HTTP (or HTTPS) communication channel. That is why there is so much footprint of HTTP on the definition of Java Servlet API package. Even we demonstrated the idea of FtpServlet by deriving it from GenericServlet for Sun Servlet API, actually, we can redefine everything from scratch, so that many FTP-related features can be much smoothly incorporated into the FtpServlet package. That would produce much flexibility for the user of FtpServlet package.

Possible WebDAV support

This section discusses the possibility to extend FtpServlet to support Web Distributed Authoring and Versioning (WebDAV)[19], which has become an important communication protocol for the Web as an extension to HTTP 1.1. It describes what WebDAV is and how the FtpServlet structure could be extended to support the feature of WebDAV. Actually this subject would be worthy a whole chapter to cover. Here we just give out some preliminary considerations about the topic.

Introduction to WebDAV

With increased focus on Internet standards and network interoperability, Web Distributed Authoring and Versioning (WebDAV) has become an important communication protocol for the Web as an extension to HTTP 1.1

HTTP 1.1 (Hypertext Transfer Protocol) has proven itself as a flexible, universal protocol for transferring data by virtue of the fact that the Web has become the bastion of the Internet. However, HTTP has some obvious shortcomings that have limited its adoption as a comprehensive Internet communication protocol. It works well for static documents intended for viewing, but does not provide the means to handle documents in a manner sophisticated enough to provide clients with rich authoring capabilities.

The current WebDAV specification tackles three major concerns of collaborative authoring tools:

Overwrite Protection.

HTTP 1.1 has no method of ensuring that clients can protect resources and make changes without fear of another client simultaneously editing them. With WebDAV, you can lock resources in a variety of ways to let other clients know you have an interest in the resource in question, or to prevent other clients from being able to access the resource.

Resource Management

HTTP deals only with direct access to an individual resource. WebDAV provides a means of organizing data more efficiently. WebDAV introduces the notion of the *collection* (analogous to a file system folder), that can contain *resources*. Resource management via WebDAV includes the ability to create, move, copy, and delete collections, as well as the ability to do the same things to the resources or files within a collection.

Document Properties

Different types of data have unique properties that help describe the data. For example, in an e-mail message, these properties might be the sender's name and time received. In a collaborative document, these properties might be the original author's name and the name of the last editor of the document. As the types of documents that people use diversify, the list of possible property types becomes very large. XML is the type of extensible communication vehicle required by WebDAV.

HTTP provides a set of methods that clients can use to communicate with servers and specifies the format of responses from servers back to the clients that have issued requests. WebDAV fully adopts all of the methods of this specification, extends some of these methods, and introduces additional methods to provide the functionality described.

XML, Transmission Format

WebDAV was designed to provide more methods for handling resources on a server. These additional methods generally require a great deal of information to be associated with both requests and responses to explicitly define the intention of the client or server. The method of communicating all of this information in HTTP was solely the responsibility of the headers in requests and responses. This imposes some limitations on transfers. It is very difficult to apply header information to multiple resources in a request and to fully represent the information due to the nature of its hierarchic structure.
Because of its inherent extensibility, XML was chosen to describe how these instructions are communicated. XML is crucial to the operation of WebDAV because it provides:

- A method of formatting instructions describing how data is to be handled.
- A method of formatting complex responses from the server.
- A method of communicating customized information about the collections and resources handled.
- A flexible vehicle for the data itself.

At a high level, a WebDAV instruction processor is really a set of logic that interprets WebDAV methods, followed by an XML parser that interprets the majority of the information communicated.

WebDAV layer built for FtpServlet

From the introduction to FTP, we have known that FTP also provides approaches to manage resources on the server-side, either uploading files to the server, or downloading files from server to the clients. Similar to what HTTP has experienced with the "lost update" limitation, as a server resource management tool, FTP also needs improvements regarding this limitation.

As we introduced in last section, webDAV provides a promising way to extend HTTP to make it suitable for server resource management. The idea of extending to webDAV can also apply to FTP. Since we have demonstrated the feasibility of FtpServlet, the addition layer of extending FTP can be implemented on top of the FtpServlet.

As shown in the figure, a WebDAV layer could be introduced between the communication channel of the web Server and the FtpServlet Engine. The additional layer is responsible for converting the request, which is in the format of a Http Request with a webDAV extension, to Ftp Request that can be understood by the FtpServlet container. On the other hand, the response from the Ftp Server will be processed by the layer to send back to the client in the format of Http Response, which is the language that a webDAV client can understand.

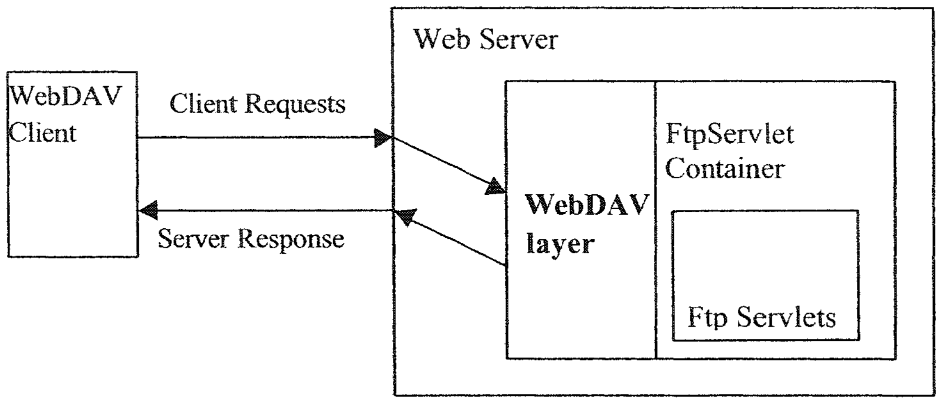

Figure 10 An additional layer could make Ftp Server support WebDAV

General Servlet Container

From the previous discussion, we know the current servlet / container paradigm strictly relies on HTTP as a protocol for requests and responses. That would impose a limitation to the applications of Servlet technology. This project has already shown the possibility to extend the servlet concept to other popular protocols.

Actually, there would be the possibility to implement a generic servlet engine, which can contain any type of servlets, such as FtpServlet, TelnetServlet, and so on. The author believes that the generic servlet container would not only give much convenience for the server development, but also provide a new mechanism for the development of other application services.

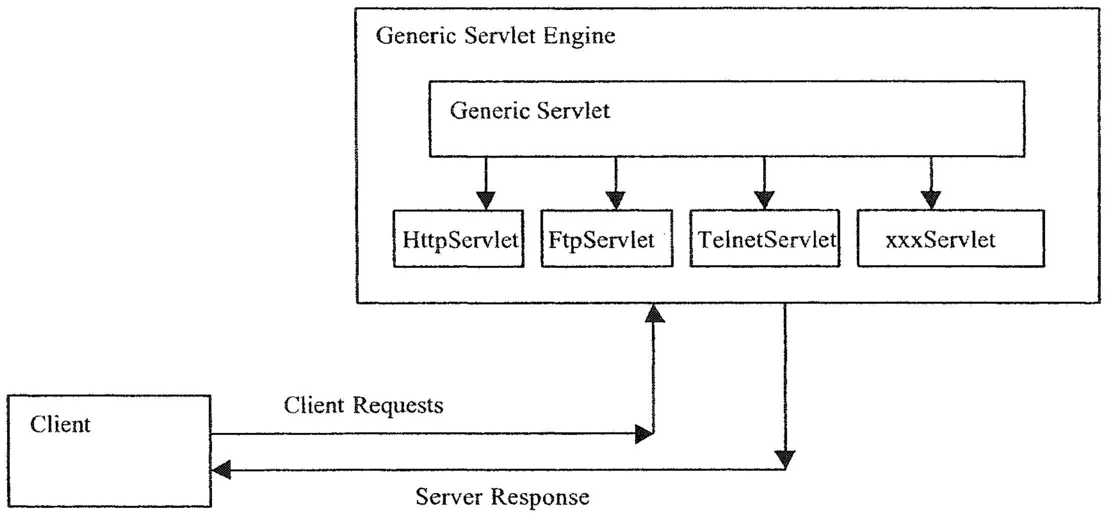

Figure 11 Possible extension plan for generic servlet engine

As depicted in the above diagram, we'd like to have a generic servlet engine to hold all kinds of servlets, and then serve any request coming from the clients. For now the client of Servlet engine always means the Internet browsers, i.e. Internet Explorer or NetScape Navigator, since all the current Servlets can only understand the requests in the format of HTTP. For any other request, the service provider must implement the server to fulfill their special requirement. For example, an FTP server must be implemented to fulfill the requirement of FTP protocol. Since now all the information is digitized and saved into a Database system, sometimes the ftp server must be acting as middle ware to serve the entire information system. The existence of a generic servlet engine and an ftp servlet would definitely simplify the development for such systems.

Bibliography

[1] Coward, D. "Java^{TM} Servlet Specification" (version 2.3), September 2001. http://jcp.org/aboutJava/communityprocess/first/jsr053/index.html

[2] Fielding, R., Gettys, J., Mogul, J.C., Frystyk, H., Masinter, L., Leach, P., and Berners-Lee, T. "Hypertext Transfer Protocol – HTTP/1.1", RFC2616, June 1999.

[3] Berners-Lee, T. and D. Connolly, "Hypertext Markup Language - 2.0", RFC 1866, November 1995.

[4] Netscape Communications Corporation, "Core JavaScript Guide", v1.4, October 1998.

[5] Lomax, P. "Learning VBScript", O'Reilly & Associates, July 1997.

[6] Coar, K.A.L,. Robinson, T. "The WWW Common Gateway Interface", Version 1.1(draft 03), June 1999.

[7] "Apache Tutorial: Introduction to Server Side Includes", http://httpd.apache.org/docs/howto/ssi.html

[8] Weissinger, A. K. "ASP in a nutshell", 2nd Edition, O'Reilly & Associates, July 2000.

[9] The Java^{TM} Web Services Tutorial http://java.sun.com/webservices/docs/1.2/tutorial/doc/index.html

[10] Postel, J. and Reynolds, J. "File Transfer Protocol (FTP)", RFC959, October 1985.

[11] Wilkins, G., What is the Jetty Architecture? http://jetty.mortbay.org/jetty/doc/JettyArchitecture.html

[12] Chang, P. I, Inside The Java Web Server, April 2003.

http://java.sun.com/features/1997/aug/jws1.html

[13] Allamaraju, S. Java Servlets: Design Issues, March 2002.

http://www.subrahmanyam.com/articles/servlets/ServletIssues.html

[14] C & K Management Limited, An Overview of Servlets

http://www.themanagementor.com/enlightenmentorareas/it/EI/AnOverview.htm

[15] Ipswitch, Inc., FTP Technical Guide

http://www.ftpplanet.com/ftpresources/ftptech.htm

[16] An Overview of the File Transfer Protocol

http://www.ncftpd.com/libncftp/doc/ftp_overview.html

[17] Peterson, B. "Understanding J2EE Application Server Class Loading Architectures", May 2002.

http://www.theserverside.com//resources/article.jsp?l=ClassLoading

[18] Fangagan, D., An Overview of the Java 2 Platform, December 1998.

http://www.oreillynet.com/pub/a/oreilly/java/news/java2_0299.html

[19] Goland, Y., Whitehead, E., Faizi, A., Carter, S. and Jensen, D. "HTTP Extensions for Distributed Authoring -- WEBDAV", RFC2518, Feburary 1999.

Appendix

Table A2 FTP commands

FTP command	**Description**
CDUP	Change to parent directory
CWD	Change working directory or library
DELE	Delete file or document
LIST	File list
MKD	Make directory
MODE	Set transfer mode
NLST	Name list
NOOP	Obtain server response
PASS	Password
PASV	Use passive data connection
PORT	Data port
PWD	Print working directory
QUIT	End an FTP server system
RETR	Retrieve file
RMD	Remove directory
RNFR	Rename from
RNTO	Rename to
STOR	Store file
STRU	Specify file structure

SYST	Identify the name of the operating system
TYPE	Specify representation type
USER	Send a user logon ID to the server
XCWD	Change working directory or library
XMKD	Make directory
XPWD	Display working directory or library
XRMD	Remove directory

Glossary

access controls

Access controls define users' access privileges to the use of a system, and to the files in that system. Access controls are necessary to prevent unauthorized or accidental use of files. It is the prerogative of a server-FTP process to invoke access controls.

client

A program that establishes connections for the purpose of sending requests.

connection

A transport layer virtual circuit established between two programs for the purpose of communication.

FTP commands

A set of commands that comprise the control information flowing from the user-FTP to the server-FTP process.

server

An application program that accepts connections in order to service requests by sending back responses. Any given program may be capable of being both a client and a server; our use of these terms refers only to the role being performed by the program for a particular connection, rather than to the program's capabilities in general.

uniform resource locator (URL)

A compact string representation of resources available via the network. Once the resource represented by a URL has been accessed, various operations may be performed on that resource. A URL is a type of uniform resource identifier (URI). URLs are typically of the form:

```
<protocol>//<servername>/<resource>
```

For the purposes of this specification, we are primarily interested in HTTP-based URLs which are of the form:

```
http[s]://<servername>[:port]/<url-path>[?<query-string>]
```

For example:

```
http://java.sun.com/products/servlet/index.html
https://javashop.sun.com/purchase
```

In HTTP-based URLs, the '/' character is reserved to separate a hierarchical path structure in the URL-path portion of the URL. The server is responsible for determining the meaning of the hierarchical structure. There is no correspondence between an URL-path and a given file system path.

web application

A collection of servlets, JSP pages, HTML documents, and other web resources that might include image files, compressed archives, and other data. A web application may be packaged into an archive or exist in an open directory structure.