The Architecture of the Domino Web server, Part 1

by Richard Schwartz

[Editor's note: This article resides in "Notes Today", the technical Webzine located on the http://www.notes.net Web site produced by Iris Associates, the developers of Domino/Notes. This is the first in a two-part series of articles about the architecture of the Domino Web Server. This first article focuses on the basic structure of the server and describes some of the behind-the-scenes mechanisms that Domino uses to respond to requests for Web pages. The second article goes into more detail about how Domino presents pages, accepts forms, performs searches, and executes agents.]

Introduction

The Lotus Domino Server contains several modules, or server tasks. One of those server tasks is the Web, or HTTP server. When you type "LOAD HTTP" on a Domino server console, a remarkable transformation occurs. Suddenly, the Domino server's world is opened up. Web developers gain access to powerful tools for interactive application development, and Web clients gain access to great applications built using technology previously available within Lotus Notes. This feature certainly has gotten a lot of attention lately, partly because the Domino Web server has capabilities that make it unlike any other Web server in existence. These capabilities make it much more than a cool way to open up existing Notes applications to Web clients -- they make it a truly powerful tool for developing new Web applications.

From the outside, Domino Web servers mostly look like all other Web servers. They have only a few distinguishing features that give away their underlying nature. One such feature, presenting "twisties" to expand or collapse hierarchical information on pages, is immediately familiar to most Macintosh users and it certainly feels natural to Notes R4 users, but it is rarely used in non-Domino Web applications. If you see twisties on the Web, more likely than not, the site is Powered by Notes. Similarly, Action buttons displayed in a row across the tops of pages and some elements of the URL language are also quite familiar to the Notes-savvy observer. These visible features, all testaments to the Notes technology environment underneath Domino, give away the fact that some really interesting things go on behind the scenes of this Web server. This article will take you behind those scenes and help you understand how the Domino 4.5a Web server works today. Please bear in mind, however, that some of the architectural information in this article, of course, is subject to change in future releases.

Domino's heritage in the Lotus Notes family of products plays a large part in its unique capabilities. This article seeks to help you understand what is going on behind the scenes in Domino, and Notes is a big part of that, so if you are already comfortable with the concepts behind Lotus Notes you should be able to read this article straight through. For readers unfamiliar with basic Notes concepts, a **glossary sidebar** is available that introduces these concepts.

Domino.Basics

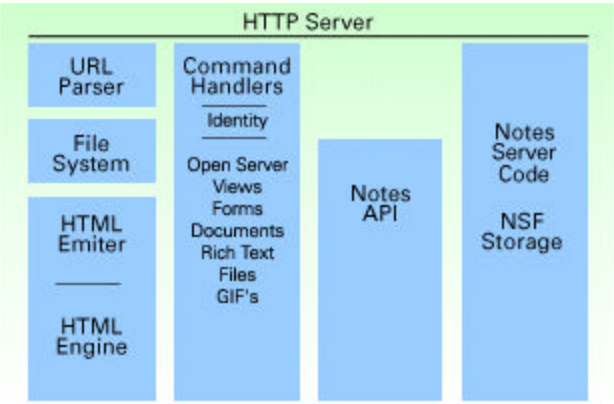
The Domino Web server is a Domino server task. It is written in C++ and it uses the Notes API to integrate itself with the rest of the Domino server in the same way that all other server add-in tasks do. For the most part, the Domino server task interacts with the rest of the Domino server the same way any 3rd party add-in program would: with the Notes API. There are a few places where it accesses Notes internals without the standard API, but never while it is reading NSF files. Its efficiency is primarily due to its clean object-oriented design, not due to any special "hooks."

The Domino Web Server is comprised of two parts: the HTTP Server and the Domino App Server. These can be thought of as the "front end" and "back end" of the overall server. The HTTP Server is a multi-threaded task that listens for Web client requests and sends responses, in much the same way that the Notes tasks in the Domino server listen for and respond to Notes client requests. That functional similarity forces a departure from "pure" Notes API programming in the Web server code. Normal server tasks run with the identity of an ID file that resides somewhere on the host file system, but in order to faithfully implement the Notes security model for Web clients it is necessary for the thread servicing a request to

assume the identity of the logged-in user. This capability exists in the Notes server core as part of its own security enforcement mechanism, but is not exposed through the Notes API, so the Domino Web server's programmers went underneath the API to accomplish this. (The reason why this capability is not exposed through the Notes API is because the Notes server must sometimes execute lookups or even scripted "agents" on behalf of users, so it essentially acts as a proxy for the user. While doing this, it assumes the identity of the user so that security will be automatically enforced. If this proxy capability were exposed through the Notes API, a programmer with access to the Notes server could create code that impersonates Web users and could gain access to private information.)

Domino.Front.End

The Domino Web server is (first but not necessarily foremost) an HTTP server. In other words, it is a TCP/IP application that implements the HyperText Transfer Protocol. It answers URL requests from client programs by sending back pages of data encoded in HyperText Markup Language (HTML). It also handles URL requests and HTML forms that trigger executable programs according to the Common Gateway Interface (CGI) specification. In these respects, the Domino Web server behaves just like any other HTTP server, responding in the standard way to standard URL requests.



In fact, the front end of the Domino Web server is a complete, industrial-strength HTTP server with all the usual facilities for accessing HTML pages stored in the file system of the host platform and running CGI programs. The front end includes all the code that deals with both inbound and outbound HTTP communications. When it receives a URL from a client, the first thing it does is pass it to the Domino URL Parser, which examines it to see if it is a plain vanilla URL, or a special Domino URL. To distinguish a Domino URL from a plain vanilla one, the parser attempts to break the URL into one, two, or three parts: an object identifier is mandatory, a command and arguments are optional. If the object identifier is the site name itself, or if the object identifier contains a Notes database name (*.nsf), it is a Domino URL. The URL Parser simply returns any plain vanilla URL back to the front end where it is handled in the traditional way. The handling of a Domino URL is more interesting.

Domino.URLs

Given the severely limited interaction that HTTP allows between a Web client and server, the Domino URL by necessity is essentially a command language for Notes applications. the URL consists of an object identifier, a command, and arguments. The object identifier specifies the host name itself, a Notes database, a view, a document, an agent, and so on.

A typical Domino URL looks like this:

http://www.xyz.com/site/app.nsf/f34a868d3e2aa2a385256345006edae6?OpenView&login

The above sample URL makes reference to a Notes view (identified by the UNID of its design document -- the long string of hexadecimal digits) in the Notes database called site/app.nsf on the www.xyz.com server. The rest of the URL, the command portion, is syntactically optional, but when it is left out, the URL Parser determines the type of the object and supplies an implicit command appropriate for that object type. The specific command in this sample URL is ?OpenView and contains an argument called &login to specify that the Domino server should force authentication of the user prior to executing the command, even if the access requirements for this view would not require it.

Domino URLs can be rather imposing to look at. This comes from the fact that when the Domino Web server dynamically creates URLs from Notes database elements, it uses UNIDs rather than human-comprehensible names. Although they appear cumbersome, UNIDs guarantee that Domino is unambiguously identifying a note. All other forms of identification are potentially ambiguous. But Domino does accept URLs without UNIDs, and as long as Domino programmers are careful to avoid ambiguities, these URLs work just as well as the URLs with UNIDs. For example, the URL **http://www.xyz.com/site/app.nsf/By+Date?OpenView&login**, which substitutes the view name "By Date" for the UNID shown in the sample URL above, is understood by the URL Parser, and it is handled in exactly the same fashion.

Domino.Performance

The URL Parser breaks a URL into different parts, performs a series of checks, provides implicit commands when necessary, and then invokes the appropriate command handlers. These modules take care of all the details associated with a specific command, establishing the correct identity for security purposes, accessing Notes databases, executing formulas and scripts, and retrieving information. Collectively, these modules, which are the core of the Domino App Server (or "back end") can be thought of as a Notes client simulator written with the Notes API and running within the Domino server. In Domino 4.5a, the command handlers use four caches to improve their efficiency: the Database Cache, the Design Cache, the HTML cache, and the Static Cache.

The Database cache holds C++ objects that the server requires in order to work with Notes databases. The most important component of these objects is the "handle" of the Notes database. A handle is a data structure used by the Notes API to identify an open database. Since users tend to repeatedly send URLs that refer to the same databases, caching these handles avoids having to repeatedly make some of the most expensive API calls.

The Design Cache is a set of hash tables, each containing the location of the Notes design elements of a database. Since design elements are used so often (in the HTML translation of every document and view) and change very infrequently, caching their location to avoid costly Notes API calls to locate them is beneficial.

There are some specific situations where caching HTML makes sense, and that is the purpose of the HTML cache. Since most Web browsers locally cache pages, there is little point in the server caching HTML that is likely to be accessed by only one user. This assumes that users don't frequently access the same server with two browsers, and this is usually a good assumption.

HTML representing a view is specific to a user if documents contain Reader fields. The Web server would have to go around the security enforced by the Notes API in order to determine that there are no documents with Reader fields that meet the selection criteria of the view. This would be terribly inefficient, so HTML for view displays is not a candidate for sharing between users. There is, however, the case of non-authenticated users. If an application allows non-authenticated access, it is likely that many different browsers will be used to access it with the same identity ("Anonymous"). All these browsers can share the HTML for views, so the HTML cache stores the output of ?OpenView commands.

The HTML generated in response to an ?OpenDatabase command may also be cached, but only in the case where ?OpenDatabase returns the default page with links to the application's views and folders. If the

launch options of the Notes database have been set to open the application's About Page, or a specific document or Navigator, then the ?OpenDatabase URL is redirected to another command handler whose output is not cached.

HTML representing a document is specific to a user if formulas are dependent on @UserName or @UserRoles, or if section security is used. Caching is also inappropriate if formulas are time-dependent. Analysis of all the formulas and script associated with a document to determine all this is simply too costly, so HTML representing documents is never cached.

The Static Cache is stored on disk rather than in RAM, and can be configured through settings in the Server document in the Public Address Book. It is referred to as "static" because it doesn't necessarily go away every time you reboot the server. This cache stores graphics and file attachments that have been extracted from Notes documents and converted to the formats expected by Web browsers. The conversion operations can be quite expensive, and in many cases the same graphics appear on many Web pages, so this cache can be a significant boost to server performance.

Domino.HTML.Generation

In order to support its true goal of hosting rich interactive Web applications that publish, capture, and process information, everything in the Domino Web server finally ends with the production of HTML to send to browsers. An intelligent HTML Emitter is used by all the command handlers. The emitter depends on a table-driven HTML Engine that has knowledge of the syntax and semantics of HTML to ensure that the output always conforms to HTML standards. For example, the emitter keeps state information so that it can ensure that termination tags are inserted wherever they are required. This reduces the complexity of the command handlers dramatically. The object-oriented code responsible for translating a view column entry or a field on a form can concentrate on its particular job without worrying about the HTML tags generated for the previous view column or for the table inside the collapsible section on the form containing the field. The result is that the on-the-fly HTML translation is remarkably efficient.

ABOUT THE AUTHOR

Richard Schwartz is founder and President of **RHS Consulting Incorporated**, a Lotus Business Partner and a member of the Penumbra Group. He is a Certified Lotus Professional, Editor of Lotus Solutions Now, contributing writer for Lotus Notes Advisor, and a frequent contributor to Notes In a Nutshell and other Notes-related publications. His activity in various Notes-related forums on the Internet led to his being honored with a Lotus Business Partners' Beacon Award in January 1995. A key contributor to Email, Directory and other network application product development at Wang Laboratories from 1983 to 1991, Rich has nearly fifteen years of hands-on experience with groupware technology.