URL Mapping

Author : *rbowen*

Date : 2002/09/2921 : 13 : 39

Contents

1 URL mapping

URL mapping is the process by which a URL is analyzed to figure out what resource it is referring to, so that that resource can be returned to the requesting client. This process is performed with every request that is made to a web server, with some of the requests being served with a file, such as an HTML document, or a gif image, others with the results of running a CGI program, and others by some other process, such as a built-in Apache handler, a PHP document, or a Java servlet.

There are a number of different things which are considered during this mapping process, and thus there are a number of ways that you can influence the process via your server configuration file.

For a treatment of some of these things, you may also wish to see the document on the Apache documentation site that deals with this topic: `http://httpd.apache.org/docs/sections.html`

2 Location

The `<Location>` directive defines a set of directives that apply to a set of URLs.

```
<Location /status>
    SetHandler server-status
</Location>
```

This directive is often used to create a resource which is managed by a handler. In the above example, requests for the URL `/status` are answered by the `server-status` handler, provided by the `mod_status` module.

Use of the `Location` directive generally implies that there is no correlation between the requested URL and any file-system entries. That is to say, there is no directory called `/status` in the `DocumentRoot` directory.

```
<Location /example>
    SetHandler perl-script
    PerlHandler Apache::Example
</Location>
```

The example above maps the URL `/example` to a `mod_perl` handler, contained in the Perl module called `Apache::Example`. As above, there is no directory `/example` in the `DocumentRoot` directory.

3 Aliases

`mod_alias` provides two directives which allow the mapping of URLs to directories outside of the `DocumentRoot` directory. These directives, `Alias` and `ScriptAlias`, provide somewhat different functions, which are described below.

Additionally, `AliasMatch` and `ScriptAliasMatch` provide regular expression matching in order to alias a larger class of URLs to particular directories.

3.1 Alias

The `Alias` directive maps a URL to a directory, which is usually located outside of the `DocumentRoot` directory. This is used when, for any reason, a directory of content is located elsewhere in your file system, and makes it unnecessary to use a symbolic link, or other technique which may compromise security.

In your default Apache configuration file, the `Alias` directive is used to map URLs to the directory of icons (`/icons/`), and to the Apache documentation (`/manual`).

```
Alias /icons/ /usr/local/apache/icons/
```

Please note that the example above is very literal. It maps the URL `/icons/` to the specified directory, but does not match the URL `/icons` - ie, without the trailing slash. Consequently, if the first argument - the URL - lacks the trailing slash, the second argument - the directory - should lack it also. In cases where there is a slash on the first argument and not on the second, you may get 404 (File Not Found) errors, with corresponding error log entries as shown below:

```
File does not exist: /usr/local/apache/iconsimage2.gif
```

If you look closely, you'll notice that there is a slash missing between `icons` and `image2.gif`, which is causing the file to not be found. This is due to the `Alias` being applied very literally.

3.2 ScriptAlias

The `ScriptAlias` directive is like an `Alias` directive with a few extra goodies. It maps a URL to a directory, and additionally indicates that everything within that directory is a CGI program, to be executed by `mod_cgi`. Thus, when any resource is requests from this URL-space, the specified file is located, and Apache will attempt to execute that file, returning whatever content is generated. In the event that the file is not an executable program or script, an error message will be returned to the client.

```
  ScriptAlias /cgi-bin/ /usr/local/apache/cgi-bin/
```

Note that the above example, which maps the URL `/cgi-bin/` to the directory `/usr/local/apache/cgi-bin/`, is roughly equivalent to the configuration section shown below:

```
Alias /cgi-bin/ /usr/local/apache/cgi-bin/
<Directory /usr/local/apache/cgi-bin/>
 Options ExecCGI
 SetHandler cgi-script
</Directory>
```

3.3 AliasMatch and ScriptAliasMatch

As with other `Match` directives, these two directives use regular expressions to match a range of arguments, rather than a single literal string.

I'll actually be talking about Regular Expressions a few sections from now, so I'll just give a single example here:

```
AliasMatch /cgi(-bin)?/ /usr/local/apache/cgi-bin/
```

The above example will create an alias for a URL of `/cgi/` and also for a URL of `/cgi-bin/`. The `?` indicates that the latter part, contained in parentheses, is optional. The parentheses, as you will see in a moment, have other functions as well.

4 Redirect

The `Redirect` directive maps one URL to another. This involves sending a 300 status code to the client, along with a `Location:` header to tell the client where to go. The client is then responsible for making a request for the indicated resource.

```
Redirect /HyperCal.html http://www.coopermcgregor.com/products/hypercal/
```

The example causes the client to redirect to the new URL when it requests the specified resource.

One can optionally specify an additional argument that modifies the exact status code that is sent to the client. See
`http://httpd.apache.org/docs/mod/mod_alias.html#redirect` for additional details on this functionality.

4.1 RedirectMatch

Because the thing that you want to redirect is often actually a set of more than one thing that you want to redirect, the `RedirectMath` directive is very useful, allowing you to specify the thing that you want to redirect as a patter, which may match one or more URLs.

To do this, it uses the regular expression library that comes with Apache. And, as promised above, we still need to tell you some things about regular expressions.

The following examples of `RedirectMatch` directives may give you a small taste of what is possible.

```
RedirectMatch [sS]upport(.*) http://www.coopermcgregor.com/support/
Redirectmatch [dD]r[Bb]acc?h?us.* http://www.drbacchus.com/
RedirectMatch (.*) https://otherserver.com$1
```

Of course, some of this may be somewhat cryptic, so we'll talk a little bit about regular expressions, and try to remove some of the mystery.

4.2 A slight tangent - Regular Expressions

A regular expression is a formula for matching a pattern. A well-crafted regular expression may match any arbitrarily complex set of strings, so that a single expression may represent dozens or hundreds of possible variants.

Perl is the language in which regular expressions have seen the richest vocabulary. Apache implements a manageable subset of the regular expression library, using something more like the `egrep` regex library, rather than the full-blown Perl one. However, in Apache 2.0, the regular expression library is the PCRE (Perl Compatible Regular Expressions) which has a substantially larger vocabulary.

Perhaps the best way to tell you about regular expressions is to jump in and give some examples. What follows is a basic vocabulary of the words used in these expressions:

```
.   Matches anything
+   "one or more"
*   "zero or more"
?   "optional"
^   Beginning of string
$   End of string
( )  Groups and/or captures
[ ]  Character class - Match one of the characters in here
^ in a character class means "not"
```

So, building on the above vocabulary, consider some of the following examples:

Zz ebra matches Zebra or zebra

- Giraff?e matches Giraffe or Girafe
- Spell?ing matches Spelling or Speling
- Spel+ing matches Speling, Spelling, Spellling, etc
- Spel*ing matches Speing, Speling, Spelling, Spelling, etc
- ab[cde]fg matches abcfg, abdfg or abefg

- ab[^c]d matches anything that starts with ab, followed by anything that is NOT c, followed by d.
- etc ...

This is just a glimpse into what you can do with regular expressions, but let's bring it back to Apache and URL mapping.

Any of the `Match` directives (`DirectoryMatch`, `AliasMatch`, `RedirectMatch`, etc.) use regular expressions to determine what they match. This allows you to have a single `RedirectMatch` directive that does the work of a dozen `Redirect` directives, for example.

With a bit of an idea what regular expressions are, we can explain some of the examples give above.

```
RedirectMatch [Ss]arah http://sarah.rcbowen.com/
```

The character class `[Ss]` matches either `S` or `s` (that is, upper-case or lower-case S. So, this one directive does the work of two `Redirect` directives, redirecting requests for either the url `sarah` or `Sarah`. (The `Redirect` directive is case-sensitive.)

Similarly, you can have a `RedirectMatch` directive which corrects for a variety of common misspellings of a particular URL

```
RedirectMatch drbacc?h?us http://www.drbacchus.com/
```

This web site, which used to be at `http://www.rcbowen.com/drbacchus/`, and is now at `http://www.drbacchus.com/` was forever misspelled by people trying to get to it. The directive above allows for one of the c's to be ommitted, and the h to be ommitted, and the URL is still valid. That is, the ? makes each of those characters optional.

Our last example is less theoretical, and very useful, using the "capturing" attribute of parentheses.

```
RedirectMatch (.*) https://other.server.com$1
```

This directive takes any request made to your server, and redirects it to that same URL, but on your secure server. This is a great directive to use if you wish to route all requests through your secure site. The URL which the user typed is preserved, with the exception of the server. For example, `http://servername/testing/index.htm` will be redirected to `https://servername/testing/index.html`.

We'll see more of this when we get to `mod_rewrite`.

4.3 RedirectTemp and RedirectPermanent

If you've ever looked at the HTTP specification, you know there's more than one way to redirect. Actually, there's 7 different redirect codes, with only a few of those in frequent use.

By default, the `Redirect` directive sends a 302 header, indicating that the redirection is temporary.

If you wish to specify a different redirection code, you can specify an additiona argument to the `Redirect` directive, as alluded to above. Or you can use the `RedirectTemp` and `RedirectPermanent` directives.

The `RedirectTemp` directive is a little silly, sending a 302 redirection header. So this is identical to using the `Redirect` directive, which also sends a 302 header.

The `RedirectPermanent` directive sends a 301 redirection header, which indicates that the redirection is permanent. This is the same as using the `Redirect permanent` syntax with the `Redirect` directive.

5 DocumentRoot

If all else fails, it must be a request for an actual document, so we look in the DocumentRoot for the path requested.

```
DocumentRoot /usr/local/apache/htdocs
```

However, if the file is not there ...

6 Error documents

By default, Apache serves up boring error pages in the event of any error condition. At the moment, we are primarily interested in 404 (Document Not Found) error documents, but error documents are served for the whole range of error conditions.

The default error documents are less than useful. They tell the user what the error was, sort of. But they don't give enough information for them to do anything about it, and generally make the user feel like the problem was their fault.

By providing a customized error message, you have the opportunity to help the user fix the problem, or at least give them links to useful (and functional) parts of your site. This is particularly useful in the event of a File Not Found error, which may mean that they mistyped a URL, or just that some other site has an invalid link to your site.

The following are examples of the `ErrorDocument` directive.

```
ErrorDocument 404 /cgi-bin/404.cgi
ErrorDocument 404 http://www.errors.com/
ErrorDocument 500 /errors/500.html
ErrorDocument 403 "You need to log in first"
```

Quotes on ErrorDocument

In Apache 1.3, when you specify a string argument to `ErrorDocument`, you start with quotes, but do not close the quotes on the end of the string. In Apache 2.0, you need to close the quotes.

As shown in the examples above, there are three possibilities of arguments passed to the `ErrorDocument` directive. You can specify a local url (such as `/errors/not_found.html`), a fully-qualified external url (such as `http://error.server.com/not_found.html`), or a string (such as ”Bad Link, Your Fault”). This last option, being the most terse of all of them, is the least desirable, and is very likely to not work with Internet Explorer.

6.1 CGI 404 error handler

One of the examples given above maps a 404 error to a CGI program. This is perhaps not a great idea, but an extended example is provided here for how you might handle 404 errors with a CGI program.

The CGI example given here, written in Perl, will send you an email message each time someone gets a 404 error on your site. Note, if you intend to implement this on your server, that this will generate a LOT of email on a normal site. You have been warned.

In your configuration file, put:

```
ErrorDocument 404 /cgi-bin/404.cgi
```

Then `/cgi-bin/404.cgi` will look like:

```
 #!/usr/bin/perl
 use Mail::Sendmail;
 use strict;

 my $message = qq~
 Document not found: $ENV{REQUEST_URI}
 Link was from: $ENV{HTTP_REFERER}
 ~;

 my %mail = (
To      => 'admin@server.com',
From    => 'website@server.com',
Subject => 'Broken link',
Message => $message,
 );
 sendmail(%mail);

 print "Content-type: text/html\n\n";
 print "Document not found. Admin has been notified";
```

Again, this is provided only as an example, not as a recommendation.

6.2 Error documents in Apache 2.0

Apache 2.0 has a new way of handling ErrorDocument that will mean much more customizable error messages, rather than the same old boring ”Document Not Found” errors.

Note that that this is not so much a new mechanism as a variety of existing things that have been carefully put together to produce this effect. It is a combination of the existing `ErrorDocument` handling, content negotiation, and server side includes, and a lot of work.

In you Apache 2.0 default configuration file, you will see the following:

```
<IfModule mod_negotiation.c>
<IfModule mod_include.c>
   Alias /error/ "@@ServerRoot@@/error/"

   <Directory "@@ServerRoot@@/error">
       AllowOverride None
       Options IncludesNoExec
       AddOutputFilter Includes html
       AddHandler type-map var
       Order allow,deny
       Allow from all
       LanguagePriority en es de fr
       ForceLanguagePriority Prefer Fallback
   </Directory>

   ErrorDocument 400 /error/HTTP_BAD_REQUEST.html.var
   ErrorDocument 401 /error/HTTP_UNAUTHORIZED.html.var
   ... etc ...
   ErrorDocument 506 /error/HTTP_VARIANT_ALSO_VARIES.html.var

</IfModule>
</IfModule>
```

In the directory `@@ServerRoot@@/error/` you will find all of those .html.var files, which contain SSI directives for building custom ErrorDocument pages. And, thanks to the efforts of several people, they are available in several languages. These error documents can be customized to your heart’s content.

The 404 page, for example, looks like the following, in English:

```
<!--#set var="TITLE" value="Object not found!" -->
<!--#include virtual="include/top.html" -->

   The requested URL was not found on this server.

 <!--#if expr="$HTTP_REFERER" -->

   The link on the
   <a href="<!--#echo encoding="url" var="HTTP_REFERER"-->">referring
   page</a> seems to be wrong or outdated. Please inform the author of
   <a href="<!--#echo encoding="url" var="HTTP_REFERER"-->">that page</a>
   about the error.

 <!--#else -->

   If you entered the URL manually please check your
   spelling and try again.

 <!--#endif -->

<!--#include virtual="include/bottom.html" -->
```

And you'll find, within that same `.var` file, translations in several other languages. Exactly how many languages you see will depend on which version of 2.0 you are using.

As you have the full array of SSI variables at your disposal, this lets you customize this page as much as you like.

Additionally, in the `include` subdirectory of the `error` directory, you will see files `top.html` and `bottom.html`, which can be customized to put these error messages within the same look as the rest of your web site, including standard navigation bars, colors, and so on. `top.html` is included at the top of each page, and `bottom.html` is loaded in the bottom.

7 Other modules that handler URL mapping

In addition to these standard configuration directives, Apache also comes with a few other modules which facilitate doing other things in the URL mapping phase. And there are a variety of other non-standard (ie don't come with Apache) modules which do URL mapping.

The ones that come with Apache are `mod_speling`, and the enormously important `mod_rewrite`.

7.1 mod_speling

First of all, yes, that is the way that the module is spelled. `mod_speling`. This was someone's idea of humor, I believe. And it makes it really hard to talk about it in a book without getting the spelling "corrected" by some well-meaning editor.

The purpose of `mod_speling`, as the name implies, is to correct spelling errors. The module is best at correcting mistyped URLs when it is a case of one letter being mistyped as another character, or two adjacent characters being transposed, or a letter being types in the wrong case (upper case vs lower case).

If you have the module installed (it is not part of a default installation if you built from source) then the feature is enabled using the directive:

```
CheckSpelling On
```

In the event of a 404 error, `mod_speling` tales over, doing a directory listing, and checking for other files in the directory that may be what you meant. In the event that more than one file is a possible match, you will receive a page which lists all possible matches. If nothing matches, then the request will be handed on down the line for `ErrorDocument` handling.

7.2 mod_rewrite

`mod_rewrite` is an enormously important module. Hopefully, you have already attended Mads Toftum's talk about `mod_rewrite`, as this does not attempt to thoroughally cover this module in all its glory. Stated very simply, `mod_rewrite` allows you to rewrite any incoming URL to anything else, using regular expressions to accomplish this.

I will provide two simple examples to illustrate this, but there is so much more than you can do that this will do no more than whet your appetite for what is possible.

7.2.1 Example 1: Ugly URLs

Our first example addresses a common problem. Due to using a CGI program we have a requirement for ugly URLs, passing several arguments to our CGI program. For example, one of our URLs might look like:

```
  http://www.server.com/cgi-bin/program.cgi?A=arg1&B=arg2
```

Although this is a rather mild example, it serves for our purposes. We would like to have a simpler URL - one that an average user might actually have a chance of remembering and typing in correctly. For example, we want the above URL to instead look like:

```
  http://www.server.com/program/arg1/arg2
```

Of course, there are several possible solutions to this problem, but we are going to use `mod_rewrite`. Here's the solution that we'll use.

```
RewriteEngine On
RewriteRule ^/program/([^/]+)/([^/]+) \
  http://www.server.com/cgi-bin/program.cgi?A=$1&B=$2
```

A few brief explanations to demystify this for those of you not familiar with regular expressions.

`[^/]+` means "one or more character that is not a slash". The square brackets indicate a character class, and the ^means "not" when it appears withing the character class.

( ) traps the results in a variable - in particular, anything appearing within a set of parentheses will later on be available in the variable $1. If there are more than one set of parentheses, the first set will produce $1, the second set $2, and so on.

The `RewriteRule` directive takes the pattern in the first argument and uses the results to build the URL in the second argument.

If that does not sufficiently demystify it, I encourage you to play with it, and you should have gone to Mads Toftum's talk.

7.2.2 Example 2: Arbitrary name-based vhosts

In our second example, we want to have every available hostname map to a virtual host on our system. That is, for any `hostname.example.com`, we want that to map to some directory, without having to actually create any virtual hosts.

The first step in this process is to create a wildcard CNAME entry in DNS - that is, an entry that will map any hostname queried to the same place.

In Bind, you can do this as follows:

```
  *   86400   IN  CNAME  www.example.com.
```

Discussion about whether this is a good idea is left up to you and your DNS administrator.

Second, we will set up the following rewrite rules:

```
  RewriteEngine on
  RewriteCond %{HTTP_HOST} ^(.+)\.example\.com$
  RewriteRule ^(.+)$ /home/%1/public_html$1
```

The `RewriteCond` directive allows you to test arbitrary environment variables, and, additionally, to use portions of them in later directives. As with the capturing behavior of parentheses described above, you can capture portions of the matched pattern. In the case of `RewriteCond`, the captured value goes into the variable %1.

As before, $1 contains the match from the RewriteRule. And, with this rule, any request coming in to the server will be mapped to a directory specified by the hostname. A request for `http://bob.example.com/index.html` will serve the file `/home/bob/public_html/index.html`

Please note that this same functionality could probably be provided using the `mod_vhost_alias` module as well.

8 mod_userdir and public_html

The final thing that we will consider is `mod_userdir` and the `UserDir` directive. This feature allows for each user on a web server system to have a personal web site running out of their home directory. Requests for a URL beginning with a ˜ (tilde) map to that user's directory. For example:

```
http://s.ms.uky.edu/˜rbowen/
```

The URL above would map to a directory defined for the user `rbowen`, and serve content out of that directory.

That was, incidentally, the URL of my first-ever web site.

Typically, this URL will map to a particular subdirectory of the users's home directory. The actual location of this directory is specified by the `UserDir` directive.

The default setting of this directive maps requests to the `public_html` subdirectory of a user's home directory

```
UserDir public_html
```

If the argument does not start with a slash, it is understood to be relative to that user's home directory. Since Windows does not have a clear idea of what a home directory is, this syntax will produce unpredictable results under Apache on Windows.

However, the alternate syntax, providing an absolute file path, will work fine on any operating system.

```
# Serve files out of /home/username/public_html
UserDir public_html

# Serve files out of somewhere else
UserDir /www/users/*/htdocs
```

In the second syntax shown above, the * will be replaced by the particular username that was provided in the URL. That is, `http://servername/ larry` will map to the directory `/www/users/larry/htdocs`.

8.1 Security caveats

There are a couple of things that you may want to consider when enabling this feature.

8.1.1 File permissions

First of all, there is the matter of file permissions. In order for this feature to work, the specified directory (usually `/home/username/public_html`) needs to be readable by the `nobody` user who is runnig Apache. This usually means that the user's home directory, and this html directory, both need to be at least 755. The execute permission is needed so that the user can get directory listings, and change into subdirectories.

Many Unix operating systems create home directories with permissions 700 on them - that is, the owner, and nobody else, may see files in the directory. This setting will have to be relaxed in order to run a web site out of this directory, and many users, understandably, will be reluctant to do this, as it will mean that other users on the system will be able to browse through their home directory. This risk can be relieved somewhat by placing confidential files in other subdirectories, and placing 700 permissions on those subdirectories. There are, of course, some files that must be in the home directory, and these files should be protected appropriately.

This risk can be removed entirely by running per-user web sites out of an entirely different directory. This can be accomplished using the alternate syntax for the `UserDir` directive, as shown in the examples above, and here:

```
UserDir /www/users/*/htdocs
```

In this scenario, you would create an entirely new directory and have users put their personal web sites in there. These directories would be readable by everyone, but the users' individual home directories can have more restrictive permissions on them. Additionally providing a symbolic link from their home directory, perhaps called `public_html`, will give them easy access, and an easy transition from other systems that they may be used to.

8.1.2 Disable for some users

Finally, you may wish to disable this feature for certain users, or, if you are even more paranoid, simply disable it for everyone, and maybe enable it for just a few users.

For example, if the user `root` has this feature, it is possible, albeit highly unlikely, that a URL such as `http://servername/ root` may grant access to the root file system. Note that a system would have to be grossly misconfigured for this to happen, but it is possible.

However, even given the unlikely nature of this exploit, we still recommend that you disable this feature for the `root` user. You may wish also to disable it for other users on your system who you just don't trust.

```
UserDir enabled
UserDir disabled root hackerdude rbowen
```

Possible bug in "UserDir enabled"

At the time of this writing, there is a bug in `mod_userdir` which causes the above syntax to fail, although the syntax is correct, and is documented this way. The bug may be fixed by the time anyone sees this paper. Simply omit the `UserDir enabled` line if you get an error when using this syntax.

And, if you just don't trust anyone, you can disable the feature entirely, or enable it for a few select trusted users.

```
UserDir disabled
UserDir enabled bob larry archibald
```

9 Summary

In summary, we've explored the phase knows as URL mapping, in which the request made by the user is translated into an actual resource that can be returned to the user, or into an appropriate error message.

This process can be influenced by a variety of configuration directives, in order to map various URLs to resources either on the file system, or handlers which produce content dynamically, or to resources on other servers entirely.