



Author: Jon Barnett
Date: 24th November 2003

URL rewriting in Apache

This technote is a quick guide to performing URL rewriting in Apache, and in particular focuses on combining this with dynamic content servers, including Tomcat and Jetty with their JK2 connectors/listeners. It will try to explain some of the intricacies of URL rewriting using the `mod_rewrite` module, and their combination with the `mod_proxy` module, and with the `mod_jk2` module. This guide assumes the use of Apache 2.0 because of the advanced capabilities of the rewrite module. It also assumes that the reader has some knowledge on the configuration of Apache.

Why rewrite URLs?

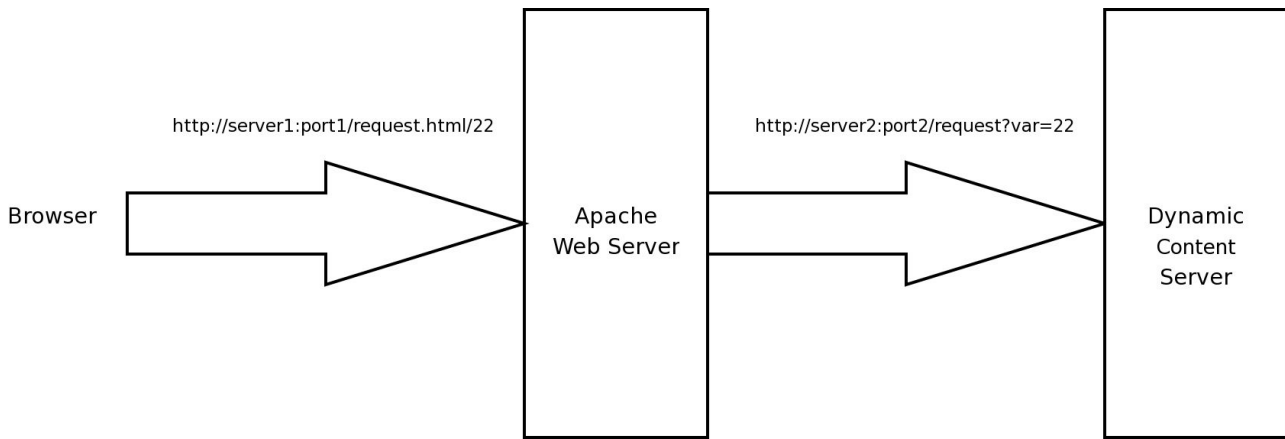
Search engines use automated agents to explore a web site to retrieve page information for indexing. The agents, also called robots, spiders or web crawlers, can be trapped by recursive links. This is achieved on web sites with dynamic content as the same page content can be referred by a link that appears to be different to one the agent has already catalogued in its search. These recursive links can be intentionally created or could arise by a mistake in the design of the dynamic site. One side effect of a dynamic site with an unintentional recursive link generator is that the trapped spider can generate page hits in rapid succession as it loops endlessly in the website search. This has the same effect as a denial of service attack. For these reasons, agents are designed to avoid dynamic pages and also tend to avoid deep searches.

Since a spider cannot detect whether a page is dynamic based on the content, the only means of detection available is to check the URL of the page. Tell-tale signs such as the key phrases `cgi-bin`, `bin`, `.jsp` and `.asp`, or the presence of query strings in the URL will lead agents to ignore the URL when crawling the site to create a content index. These pages will then be excluded from the search engine's list.

Rewriting in Apache

So in order to allow a dynamic web site to be indexed by search engines, the URL references within a web site must be made to look as though they are references to static content. However, the dynamic content server must still be able to interpret the content request.

Apache's rewrite module is one way of achieving this goal. It is able to rewrite a URL into a new form and then forward the rewritten URL to another server or handle the rewritten request itself. Forwarding the request can be achieved by the proxy module or the JK2 module, depending on the dynamic content server to which you are forwarding the requests for content.



The rewrite module in Apache 2.0 is rules based and provides Perl-type pattern matching and replacement (regular expressions). The trade-off for adding this dynamic URL obfuscation is additional processing in the rewriting stage. The delay is dependent on the amount of rewriting that is required. The more complex the translation the greater the number of rules required to transform the URL and therefore the greater the delay in serving content.

The `mod_proxy` module for Apache simply forwards the request and Apache acts as a proxy server between the content requester (browser) and the destination content server. You can also use the `mod_jk2` module to forward requests to a Tomcat/Jetty web application server using the high-performance JK2 connector.

Building the modules

In order to obtain the modules, it is usually required that you build your own version of Apache rather than use the vanilla Apache web server supplied with your UNIX/Linux distribution. Since my builds are usually for production sites, I also include the SSL module in the builds. SSL communication is performed faster in a native code web server rather than in a Java bytecode system such as Tomcat or Jetty. You also need to compile with Dynamic Shared Object support (DSO) so modules can be loaded at run-time rather than statically linked into Apache. The following shows a build process that will create an Apache installation in `/usr/local/apache2`, when compiling from the Apache 2.0 source code.

```

./configure --enable-proxy=shared --enable-rewrite=shared \
  --enable-ssl=shared --enable-so
make
make install
  
```

The JK2 module needs to be compiled separately as it is obtained from the Tomcat project. For instructions on how to compile, install and use `mod_jk2`, refer to <http://www.amitysolutions.com.au/documents/JK2-technote.pdf>.

You will only need the `mod_jk2` module and the proxy module if you intend connecting to a Tomcat or Jetty web application server via the JK2 connector. You will only need the `mod_proxy` module if you intend forwarding the request to another content server. Due to the current operation of Apache, you will need the proxy module when running complex replacements and intend to use the JK2 module. However, it does not matter if you compile all the modules as these will be dynamically loaded. That is; they will only be used when we specifically refer to them in the configuration.

For the remainder of this technical note, we will refer to the root installation directory of Apache as `APACHE_HOME`.

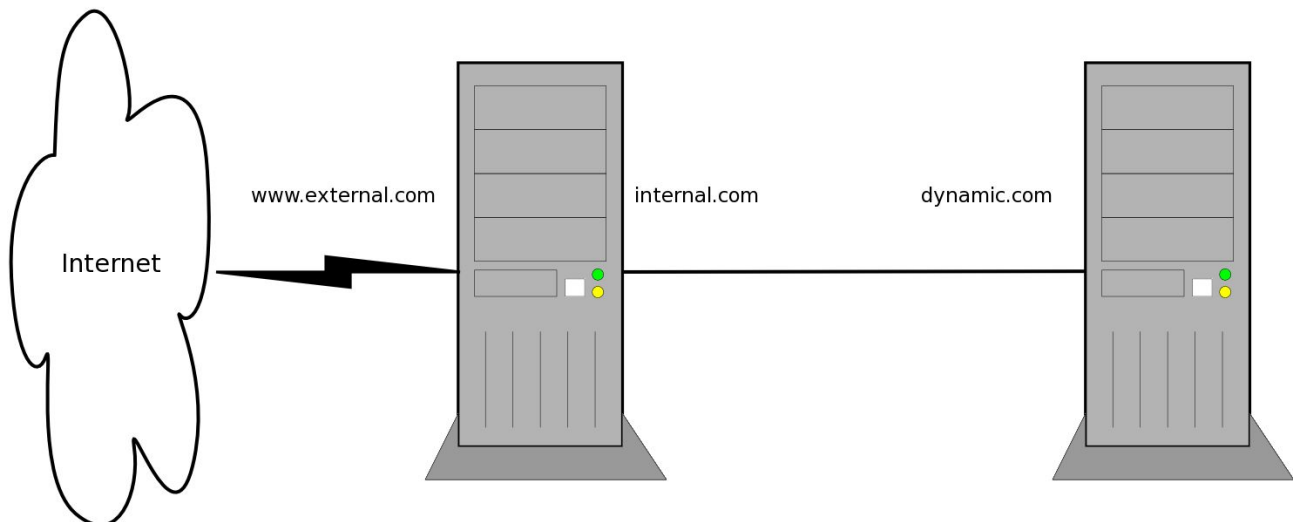
The complete set of modules as required for the URL rewriting task will have been installed in /usr/local/apache2/modules and should be as follows.

- mod_jk2.so
- mod_proxy_connect.so
- mod_proxy_ftp.so
- mod_proxy_http.so
- mod_proxy.so
- mod_rewrite.so
- mod_ssl.so

You can use your existing Apache configuration file or create a new one. I use the high-performance configuration template as it is minimal and could compensate for the URL rewriting lag.

Example scenario

The example I give adds a few complexities to show some possible real-world issues with which you may have to deal. Suppose we have a dual-homed server for Apache and it forwards the request to an internal dynamic content server.



We will use a simple URL replacement example as we aren't giving a tutorial on regular expressions. Suppose we have a URL like this:

```
http://www.external.com/documents/main;category,browse;selection,1;source,list
```

You want to convert it to this:

```
http://dynamic.com/documents/main?category=browse&selection=1&source=list
```

We'll cover the two cases where the first will describe the use of the proxy module and the second will describe the use of the JK2 module. We'll also address the difference between internal and external access.

Using rewrite and proxy

In order for Apache 2.0 to use the rewrite and proxy modules, you will need to load the dynamic modules. Assuming that our configuration supports HTTP and HTTPS, the start of our `APACHE_HOME/conf/httpd.conf` file would look like this:

```
LoadModule proxy_module modules/mod_proxy.so
LoadModule proxy_http_module modules/mod_proxy_http.so
LoadModule rewrite_module modules/mod_rewrite.so
LoadModule ssl_module modules/mod_ssl.so

Listen 80
Listen 443

ServerRoot /usr/local/apache2
DocumentRoot /usr/local/apache2/htdocs
```

For a dual-homed server, Apache 2.0.48 or earlier has some problems with rewriting and inserting the correct server name. By default, it uses the name associated with the primary interface. This may cause problems for external users accessing your website as a URL rewrite could cause the internal or private name of server to be written into the URL.

In order to prevent this, you will need to create virtual host configurations for each interface of your dual-homed server. Since the Apache instance is a proxy for an internal dynamic content server, you are going to have to route proxy requests to the server labeled `dynamic.com`. This server accepts page requests on port 8080 and you want it to send its responses back through Apache to the client browser. The external interface of the Apache server is `www.external.com`. Your virtual host section for `www.external.com` would contain the following lines:

```
<Virtualhost www.external.com>
  ProxyPassReverse / http://dynamic.com:8080/
  ServerName www.external.com
  ...
</Virtualhost>
```

This configuration defines the interface it serves (`www.external.com`), the proxy target (`http://dynamic.com:8080/`) and the server name that will be used in the rewriting (`www.external.com`). The explicit server name ensures that this is used instead of a default server name. The `ProxyPassReverse` defines that the Apache server will act as the masquerading proxy between the server, `dynamic.com` and the browser.

For similar reasons, you will want to do the same for the internal interface so that internal users have the URL correctly rewritten with the name of the internal interface.

```
<Virtualhost internal.com>
  ProxyPassReverse / http://dynamic.com:8080/
  ServerName internal.com
  ...
</Virtualhost>
```

We also need to switch on the rewriting engine, and add logging as necessary. For best performance, turn off the rewriting logging after you have finished debugging your rewriting tests by setting the `RewriteLogLevel` to 0. The lines added to the virtualhost section for the external interface are:

```
RewriteEngine on
RewriteLogLevel 0
RewriteLog logs/external_rewrite_log
```

You can do the same for the internal interface with the following:

```
RewriteEngine on
RewriteLogLevel 0
RewriteLog logs/internal_rewrite_log
```

The logs will appear in `APACHE_HOME/logs`. Set the `RewriteLogLevel` to 5 or greater for information for determining the rule operation. Do not remove the line altogether as this will not prevent the creation of the rewrite logs. Only setting `RewriteLogLevel` to 0 disables the writing of log information.

Having completed the preliminary configuration, we can now turn our attention to the problem of rewriting. Rewriting rules work in a sequential manner. The rules act in the following way:

- Test the path section of the URL
- If it matches, execute the replacement
- If it matches, perform the optional action after replacement - this may result in the restart of the processing of the rewrite rules, forwarding of the request with the rewritten URL or the skipping rules

First, most content servers would normally rewrite requests for improperly completed URLs. For example, `http://www.external.com` and `http://www.external.com/` would be redirected to `http://www.external.com/index.html`. The dynamic content server cannot do that now it is behind a proxy server. So, you need some rewriting rules that will deal with the problem. The following rules accomplish this:

```
# Replace with index.html if the URL ends with '/'
RewriteRule ^/(.+)/$ /$1/index.html [R]
# Replace with index.html if we don't find a '.' or a query
RewriteRule ^/([^?=&#;\.]+)$ /$1/index.html [R]
# Replace with /index.html if we find just '/'
RewriteRule ^/$ /index.html [R]
```

The '[R]' indicates that we redirect the browser to this address and the browser's displayed URL is rewritten to the new form. It also means that the rule processing is terminated because there was a match and we have a redirect action. The new URL is submitted to Apache and the processing rules are re-applied as it appears to the processing engine that this is a new request.

Note that we ensure if we encounter a normal query or our expected new query form, we don't append the '/index.html'. We assume with the rewrite rule that if a '.' does occur in the substring after the final '/', then it is referring to a file target, not a directory target. This covers static content that have a file extension. Also note that the order is important. We check for a terminating '/' first to append index.html. Otherwise if the third rewrite rule were applied first, we would have a double '/' implemented in the new URL.

So the URLs following would have /index.html appended and the browser would be redirected to the new URL:

```
http://www.external.com/content
http://www.external.com/content/
```

The following URLs would not have /index.html appended:

```
http://www.external.com/content/test.html
http://www.external.com/content/main;var,1
http://www.external.com/content/other#link
http://www.external.com/content/main?var=1
http://www.external.com/content.old/main
```

The last shows perhaps an erroneous format. One of the pitfalls with the rewriting engine is that you have to be careful about the way you structure your dynamic applications and content.

The last part of the task is to rewrite URLs from the search-engine friendly form to a proper query form. In our case, this is fairly simple with the following lines doing the job.

```
# Replace ',' with '='
RewriteRule ^/(.+),(.+)$ /$1=$2 [N]
# Replace ';' after first ';' with '&'
RewriteRule ^/(.+);(.+);(.+)$ /$1;$2&$3 [N]
# Replace ' ' with '%20'
RewriteRule ^/(.+)\ (.+)$ /$1\%20$2 [N]
# Replace first ';' with '?'
RewriteRule ^/(.+);(.+)$ /$1?$2
```

The [N] signals that if this rule is executed, to go back to the start of the rewriting rules for this section and start again, using the result of this rewriting rule for matching with the patterns. This is necessary so we recursively replace the symbols with the correct query symbols.

To deal with spaces in query variables, we also have to add in a replacement to convert these to %20 form. The last rule in this list has no final target instruction, which means that the rule following this is then executed. Also, the results of these rules are not reflected on the client browser, unlike the [R] instruction.

The final rule and action is to send the resultant URL request through to Tomcat. Rather than using the standard ProxyPass directive, which bypasses any rewriting rules, we use the RewriteRule as follows:

```
# Now send everything through to proxy
RewriteRule ^/(.+)$ http://dynamic.com:8080/$1 [P]
```

The [P] indicates the resulting rewrite is the request sent to the dynamic content server. This completes the objectives we needed to meet. These rules send all requests to the dynamic content server, even requests for static content. You can modify this so that static content is handled locally as normal for the Apache web server. The modification would be to remove the last line and modify the line that inserts the '?'.

```
# Replace first ';' with '?' and send to proxy
RewriteRule ^/(.+);(.+)$ http://dynamic.com:8080/$1?$2 [P]
```

Hence every URL that did not have a query string and was not a directory reference will be handled by the content definition for this virtual host, using the normal Apache Directory configurations. The conditions for standard content handling is that no rules must have been matched in the current request processing for the rewrite engine. The [N] directive does not reset the current request processing. The [R] directive and the [P] directive when Apache proxies to itself, do reset the current request processing as these appear as new content requests to the Apache server.

A complete configuration would appear as shown below.

```
<VirtualHost www.external.com:80>
    ProxyPassReverse / http://dynamic.com:8080/
    ServerName www.external.com

    RewriteEngine on
    RewriteLogLevel 0
    RewriteLog logs/external_rewrite_log

    # Replace with index.html if the URL ends with '/'
    RewriteRule ^/(.+)/$ /$1/index.html [R]
    # Replace with index.html if we don't find a '.' or a query
    RewriteRule ^/([^?=&;,#\.]*)$ /$1/index.html [R]
    # Replace with /index.html if we find just '/'
    RewriteRule ^/$ /index.html [R]
    # Replace ',' with '='
    RewriteRule ^/(.+),(.+)$ /$1=$2 [N]
    # Replace ';' after first ';' with '&'
    RewriteRule ^/(.+);(.+);(.+)$ /$1;$2&$3 [N]
    # Replace ' ' with '%20'
    RewriteRule ^/(.+)\ (.+)$ /$1\%20$2 [N]
    # Replace first ';' with '?'
    RewriteRule ^/(.+);(.+)$ /$1?$2
    # Now send everything through to proxy
    RewriteRule ^/(.+)$ http://dynamic.com:8080/$1 [P]
</VirtualHost>
```

One important point to note is that the proxy directive [P] does not result in the rewritten URL being visible to the requester. Unlike the [R] directive, the browser will not be updated with the rewritten URL.

As can be seen, the rewriting form using regular expressions is simple but understanding the flow of processing can be difficult. We would suggest that you study the application of regular expressions with either Perl reference manuals or online references, as well as using the Apache document on URL rewriting, <http://httpd.apache.org/docs-2.0/misc/rewriteguide.html> in order to understand the operation of the rules shown here. The aim we have here is to demonstrate the principles of rewriting and redirecting URL content requests. The complexity of regular expression processing is a sufficiently large topic on its own that we will not attempt to reproduce here.

Using rewrite and JK2

Working with the JK2 module is a simple extension of the standard JK2 operation. However, the constraints of the rewriting engine mean that the rewritten request must be forwarded to the Apache web server so that the JK2 system can forward the request to the Tomcat or Jetty web application server. This is necessary since, as before with the case of handling static content locally, the JK2 redirection is ignored if the rewriting module has been activated by the current content request.

We use the proxy module to forward the rewritten request to Apache so that the JK2 module can in turn forward it to the web application server. Configure the JK2 forwarding rules in workers2.conf and your web application server as necessary, based on the general directions in <http://www.amityolutions.com.au/documents/JK2-technote.pdf> or as per any JK2 instructions with which you are comfortable.

The necessary modules for loading in Apache 2.0, given our incremental change to conditions are defined in APACHE_HOME/conf/httpd.conf now as:

```
LoadModule proxy_module modules/mod_proxy.so
LoadModule proxy_http_module modules/mod_proxy_http.so
LoadModule rewrite_module modules/mod_rewrite.so
LoadModule jk2_module modules/mod_jk2.so
LoadModule ssl_module modules/mod_ssl.so
```

The only changes necessary are to remove the ReverseProxyPass line and remove the explicit proxy rewrite to target the dynamic server. The proxying in this case has no masquerading as the JK2 module ultimately handles the proxying implicitly. The proxying carried out by the web server is back to itself. As in the previous case, the browser does not directly communicate with the dynamic content server.

Refer to http://httpd.apache.org/docs-2.0/mod/mod_rewrite.html for more information on the details for creating rewriting rules.

```
<VirtualHost www.external.com:80>
    ServerName www.external.com

    RewriteEngine on
    RewriteLogLevel 0
    RewriteLog logs/external_rewrite_log

    # If no rewrite rules are matched, then JK2
    # is used

    # Replace with index.html if end with '/'
    RewriteRule ^/(.+)/$ /$1/index.html [R]
    # Replace with /index.html if we find just '/'
    RewriteRule ^/$ /index.html [R]
    # Replace ',' with '='
    RewriteRule ^/(.+),(.+)$ /$1=$2 [N]
    # Replace ';' after ';' with '&'
    RewriteRule ^/(.+);(.+);(.+)$ /$1;$2&$3 [N]
    # Replace ' ' with '%20'
    RewriteRule ^/(.+)\ (.+)$ /$1\%20$2 [N]
    # Replace first ';' with '?' and force proxy
    RewriteRule ^/(.+);(.+)$ /$1?$2 [P]
</VirtualHost>
```

There is a trap for the uninitiated here. You must use the server name that exactly matches the server name the external users employ in the URL. You cannot put server.external.com if the web users do not use http://server.external.com as the URL.

So the actions for this Apache service configuration will be:

1. Rewrite the URL where necessary, and if rewritten, send the result back to itself as a new request
2. If no rewriting is performed for this URL request, apply the JK2 redirection rules
3. If there are no matching JK2 redirection rules, handle the request as a request for local static content

This completes the basic working example for URL rewriting. The Apache 2.0 documentation contains other rewriting examples but the examples covered here relate specifically to dynamic content systems.

In terms of your dynamic content servers, the Content Management Systems must be able to generate link references that are web crawler friendly. With custom applications, this is a simple task for the developer, particularly when using Java Server Pages (JSP) or servlets based applications. Once you have determined the construction of your links, you can work through the rewriting rules required to transform the URLs back to normal form dynamic page queries, using the examples here as a reference.

No doubt there are other ways to combine URL rewriting with dynamic content servers or achieve the same effect of allowing web crawlers to explore and catalogue a dynamic site. The example detailed here provides one method that uses supported functionality of the Apache web server, and can integrate with the mod_jk2 connector for Tomcat and Jetty.