RewritePathInfo

Recipes found here map PATH_INFO arguments to QUERY_STRING arguments. This will start with simple examples, and move to more complex ones.

Moving path information to a query string - simple

Problem:

Map URLs of the form:

```
http://servername/example/arg
```

to URLs of the form

```
http://servername/something.php?arg
```

Recipe:

```
RewriteEngine On
RewriteRule ^/example/(.*) /something.php?$1 [PT]
```

Discussion:

As the simplest possible example of this rule, there are a many more specific cases in which this will fail. However, as a starting place it's pretty good.

Potential problems include Looping. (ie, when the target of the rule matches the original condition, causing the rule to run again, and again, ad infinitum.) See also the more complex cases below for when you have more than one argument which you wish to rewrite.

Moving path information to a query string - intermediate

Problem:

Map URLs of the form:

```
    http://servername/example/one/two
```

to URLs of the form

```
    http://servername/something.cgi?arg=one&other=two
```

Recipe:

```
    RewriteEngine On
    RewriteRule ^/example/([^/]*)/([^/]*) /something.cgi?arg=$1&other=$2 [PT]
```

Discussion:

When you want to rewrite more complex URLs, you need to create more complex regular expressions. Just about any pattern can be expressed as a regular expression, if you break it down into small chunks. This regular expression breaks down into just a few component parts, once you get past staring at the seemingly random characters:

```
    [^/]
```

The above component is a character class containing a "not slash". So, if we do ...

```
[^/]*
```

... that means "zero or more not-slash characters". In other words, we're looking for everything between the slashes. There are two sets of these, because we're looking for two blocks of things between slashes. Armed with that little nugget of information, go look at the regular expression again and see if it makes a little more sense.

As with the earlier recipe, I used the [PT] flag to indicate that the target URL was not merely a file to be served, but was something that needed to be handled. In this case, it's going to be a cgi-script handler. So Apache passes the resulting URL through to that handler.

Moving path information to a query string - Advanced

Problem:

Map URLs of the form

```
/blah/
/blah/one/
/blah/one/two/
/blah/one/two/three/
etc...
```

to

```
blah.php
blah.php?arg1=one
blah.php?arg1=one&arg2=two
blah.php?arg1=one&arg2=two&arg3=three
etc...
```

all with one RewriteRule.

Recipe

```
RewriteEngine On
RewriteRule ^/blah/?([^/]*)/?([^/]*)/?([^/]*)/?([^/]*)/?  \
 /blah.php?arg1=$1&arg2=$2&arg3=$3&arg4=$4 [PT]
```

Discussion

This recipe really deals more with knowledge of how PHP and other server-side languages handle their arguments than how mod_rewrite works, but it is useful to save yourself work with RewriteRules. There is really only one thing making this recipe different from the one above: the inclusion of a question mark (*?*) after every slash (*/*) (except the first slash). This means that each slash is optional, or more specifically, there is either one slash or no slashes – but not more than one – in each position marked by */?*. This works well when you have one script that handles a lot of stuff.

I'll illustrate by way of an example. Say you're navigating a book store site, and the main store browsing script is *store.php*. When you first start to browse, you see *store.php* with no arguments passed to it, so the script shows you a list of categories of books. You click on one of those categories and are taken to *store.php?category=computer*. Within this category, you can browse by title, or search by title or author. If you click "browse" then you're taken to a list of the letters of the alphabet, one of which you click on (say "B"), thus taking you to a page with books beginning with that letter. There are of course many books beginning with that letter, and results are only displayed 10 per page, so you can choose a page number to take you to a different page (say "7"). By this time, you're at

```
/store.php?category=computer&action=browse&argument=b&page=7
```

Or, in our rewritten world,

```
/store/computer/browse/b/7/
```

This would be achieved with a RewriteRule of:

```
RewriteRule ^/store/?([^/]*)/?([^/]*)/?([^/]*)/?([^/]*)/?  \
 /store.php?category=$1&action=$2&argument=$3&page=$4    [PT]
```

Note that at each level, the script still provides you with the relevant information. Thus you proceed from */store/* to */store/computer/* to */store/computer /browse/* to */store/computer/browse/b/* to */store/computer/browse/b/7/* all with the same RewriteRule.

Now, back to what I said at the beginning of this recipe. What happens when a viewer points his browser to */store/*? That is rewritten to the script GET query */store.php?category=&action=&argument=&page=*, which you (or your web programmer) should handle appropriately with your choice of CGI languages. For example, in PHP,

```
( isset($_GET['category']) && $_GET['category']!='' )
```

would return *FALSE* with this GET string. The boolean expression above is what should be used anyway, before mod_rewrite comes into play.

Also note that a side benefit of this RewriteRule is that trailing slashes aren't required, i.e. */store/computer/browse/* == */store/computer/browse* (because of the _/?_s). If you have users often typing in URLs, this could prove handy, but your web programmer should be careful with relative links in his pages.

A further pitfall to avoid is taking what I said too literally. "Slashes are optional" doesn't mean that *storecomputerbrowseb7* will work; rather, the string *storec omputerbrowseb7* will be treated as $1, the "category" argument to the script (which, if your script is particularly written, could be handled okay if you wanted it to be). If you will only have certain things in certain places, you could specify them, and then the slashes would be *truly* optional, although this is likely of limited usefulness, although the concept is good to illustrate. For example, if "category" only accepted values of "computer" and "scifi", and "action" only accepted arguments of "browse" and "search", and "argument" only took single letters, and "page" only took integers, your RewriteRule could look like this:

```
RewriteRule ^/store/?(computer|scifi)?/?(browse|search)?/?([A-Za-z]?)/?([0-9]*)/?  \
 /store.php?category=$1&action=$2&argument=$3&page=$4    [PT]
```

Note that all this was doing was replacing all of the *([^/]*)_s with things like _(browse|search)*, meaning that the regexp would only match exactly either "browse" or "search" in this position, but not anything else. Note that, with this RewriteRule, it is perfectly legal for any particular part to be completely left out, e.g. */store/browse/2/* is rewritten to */store.php?category=&action=browse&argument=&page=2*, which likely would cause your script to provide some crazy results. For this problem not to occur, you'll have to use [LookAheads] and [LookBehinds] and/or [RewriteConds], which bump up the complexity beyond my level of understanding. Hopefully [DrBacchus] will enlighten us. *BTW DrB: Feel free to edit, throw away, whatever. You're the experienced author here, and I'm just a kid who doesn't know how to divide complex topics into managable and meaningful chunks.*

And this *should* work, as well as potentially saving some CPU time for normal URLs (with slashes) since the regexp is only looking for certain things in certain places, as opposed to mostly wildcards.

Further note that the query arguments MUST come in the same order every time. This works in the store example, because a page number does not make sense unless the category, action, and argument are specified; the (browse/search) argument doesn't make sense until the category and action are specified, and so on. If you want to have query arguments in arbitrary order, you will need another recipe.

Note: Other recipe involves /?([^/])/? ... and expects things like /category=computer/action=browse/ etc. and has a limit of 9 arguments. Or, alternatively, /category/computer/action/browse, but then you're limited to 4 arguments.*