Nginx

Nginx location:

Nginx location block section have a search order, a modifier, an implicit match type and an implicit switch to whether stop the search on match or not. the following array describe it for regex.

```
# ------------------------------------------------------------------------
------------------------------------------------------------------
# Search-Order  Modifier  Description
Match-Type        Stops-search-on-match
# ------------------------------------------------------------------------
------------------------------------------------------------------
#     1st          =      The URI must match the specified pattern exactly
Simple-string              Yes
#     2nd          ^~     The URI must begin with the specified pattern
Simple-string              Yes
#     3rd        (None)   The URI must begin with the specified pattern
Simple-string                No
#     4th          ~      The URI must be a case-sensitive match to the
specified Rx      Perl-Compatible-Rx   Yes(first match)
#     4th          ~*     The URI must be a case-insensitive match to the
specified Rx    Perl-Compatible-Rx   Yes(first match)
#     N/A          @      Defines a named location block.
Simple-string              Yes
# ------------------------------------------------------------------------
------------------------------------------------------------------
```

Capturing group:

Capturing group, expression evaluation () are supported, this example location ~ ^/(?:index|update)$ match url ending with example.com/index and example.com/update

```
# ------------------------------------------------------------------------
---------------
#    ()    : Group/Capturing-group, capturing mean match and
retain/output/use what matched
#            the patern inside (). the default bracket mode is "capturing
group" while (?:)
#            is a non capturing group. example (?:a|b) match a or b in a non
capturing mode
# ------------------------------------------------------------------------
---------------
#    ?:    : Non capturing group
#    ?=    : Positive look ahead
```

```
#    ?!    : is for negative look ahead (do not match the following...)
#    ?<=   : is for positive look behind
#    ?<!   : is for negative look behind
# ------------------------------------------------------------------------
---------------
```

The forward slash:

Not to confuse with the regex slash \, In nginx the forward slash / is used to match any sub location including none example location /. In the context of regex support the following explanation apply

```
# ------------------------------------------------------------------------
---------------
#     /    : It doesn't actually do anything. In Javascript, Perl and some
other languages,
#            it is used as a delimiter character explicitly for regular
expressions.
#            Some languages like PHP use it as a delimiter inside a string,
#            with additional options passed at the end, just like Javascript
and Perl.
#            Nginx does not use delimiter, / can be escaped with \/ for code
portability
#            purpose BUT this is not required for nginx / are handled
literally
#            (don't have other meaning than /)
# ------------------------------------------------------------------------
---------------
```

The slash:

The first purpose of the regex special character \ is meant to escape the next character; But note that in most case \ followed by a character have a different meaning, a complete list is available here.

Nginx does not require escaping the forward slash / it does not either deny escaping it like we could escape any other character. and thus \/ is translated/matching /. One purpose of escaping forward slashes in the context of nginx could be for code portability.

Other regex chars

Here is a non exhaustive list of regex expression that can be used

```
# ------------------------------------------------------------------------
---------------
#     ~     : Enable regex mode for location (in regex ~ mean case-sensitive
match)
```

```
#     ~*    : case-insensitive match
#     |     : Or
#     ()    : Match group or evaluate the content of ()
#     $     : the expression must be at the end of the evaluated text
#             (no char/text after the match) $ is usually used at the end of
a regex
#             location expression.
#     ?     : Check for zero or one occurrence of the previous char ex jpe?g
#     ^~    : The match must be at the beginning of the text, note that
nginx will not perform
#             any further regular expression match even if an other match is
available
#             (check the table above); ^ indicate that the match must be at
the start of
#             the uri text, while ~ indicates a regular expression match
mode.
#             example (location ^~ /realestate/.*)
#             Nginx evaluation exactly this as don't check regexp locations
if this
#             location is longest prefix match.
#     =     : Exact match, no sub folders (location = /)
#     ^     : Match the beginning of the text (opposite of $). By itself, ^
is a
#             shortcut for all paths (since they all have a beginning).
#     .*    : Match zero, one or more occurrence of any char
#     \     : Escape the next char
#     .     : Any char
#     *     : Match zero, one or more occurrence of the previous char
#     !     : Not (negative look ahead)
#     {}    : Match a specific number of occurrence ex. [0-9]{3} match 342
but not 32
#             {2,4} match length of 2, 3 and 4
#     +     : Match one or more occurrence of the previous char
#     []    : Match any char inside
# ------------------------------------------------------------------------
------------------
```

When Location Block Evaluation Jumps to Other Locations

When a location block is selected to serve a request, the request is handled entirely within that context from that point onward. However, it is important to realize that there are times when a new location search is triggered by certain directives within the selected location. The exceptions to the “only one location block” rule may have implications on how the request is actually served and may not align with the expectations you had when designing your location blocks. Some directives that can lead to this type of internal redirect are:

- index
- try_files
- rewrite

- error_page

index directive

Always leads to an internal redirect if it is used to handle the request. Exact location matches are often used to speed up the selection process by immediately ending the execution of the algorithm. However, if you make an exact location match that is a directory, there is a good chance that the request will be redirected to a different location for actual processing.

In this example, the first location is matched by a request URI of /exact, but in order to handle the request, the index directive inherited by the block initiates an internal redirect to the second block:

```
index index.html;

location = /exact {
    . . .
}

location / {
    . . .
}
```

In the case above, if you really need the execution to stay in the first block, you will have to come up with a different method of satisfying the request to the directory. For instance, you could set an invalid index for that block and turn on autoindex:

```
location = /exact {
    index nothing_will_match;
    autoindex on;
}

location  / {
    . . .
}
```

This is one way of preventing an index from switching contexts, but it’s probably not useful for most configurations. Mostly an exact match on directories can be helpful for things like rewriting the request (which also results in a new location search).

try_files directive

Another instance where the processing location may be reevaluated is with the try_files directive. This directive tells Nginx to check for the existence of a named set of files or directories. The last parameter can be a URI that Nginx will make an internal redirect to.

Consider the following configuration:

```
root /var/www/main;
```

```
location / {
    try_files $uri $uri.html $uri/ /fallback/index.html;
}

location /fallback {
    root /var/www/another;
}
```

In the above example, if a request is made for /blahblah, the first location will initially get the request. It will try to find a file called blahblah in /var/www/main directory. If it cannot find one, it will follow up by searching for a file called blahblah.html. It will then try to see if there is a directory called blahblah/ within the /var/www/main directory. Failing all of these attempts, it will redirect to /fallback/index.html. This will trigger another location search that will be caught by the second location block. This will serve the file /var/www/another/fallback/index.html.

rewrite directive

Another directive that can lead to a location block pass off is the rewrite directive. When using the last parameter with the rewrite directive, or when using no parameter at all, Nginx will search for a new matching location based on the results of the rewrite.

For example, if we modify the last example to include a rewrite, we can see that the request is sometimes passed directly to the second location without relying on the try_files directive:

```
root /var/www/main;
location / {
    rewrite ^/rewriteme/(.*)$ /$1 last;
    try_files $uri $uri.html $uri/ /fallback/index.html;
}

location /fallback {
    root /var/www/another;
}
```

In the above example, a request for /rewriteme/hello will be handled initially by the first location block. It will be rewritten to /hello and a location will be searched. In this case, it will match the first location again and be processed by the try_files as usual, maybe kicking back to /fallback/index.html if nothing is found (using the try_files internal redirect we discussed above).

However, if a request is made for /rewriteme/fallback/hello, the first block again will match. The rewrite be applied again, this time resulting in /fallback/hello. The request will then be served out of the second location block.

A related situation happens with the return directive when sending the 301 or 302 status codes. The difference in this case is that it results in an entirely new request in the form of an externally visible redirect. This same situation can occur with the rewrite directive when using the redirect or permanent flags. However, these location searches shouldn’t be unexpected, since externally visible redirects always result in a new request.

Error_page directive

Can lead to an internal redirect similar to that created by try_files. This directive is used to define what should happen when certain status codes are encountered. This will likely never be executed if try_files is set, since that directive handles the entire life cycle of a request. Consider this example:

```
root /var/www/main;

location / {
    error_page 404 /another/whoops.html;
}

location /another {
    root /var/www;
}
```

Every request (other than those starting with /another) will be handled by the first block, which will serve files out of /var/www/main. However, if a file is not found (a 404 status), an internal redirect to /another/whoops.html will occur, leading to a new location search that will eventually land on the second block. This file will be served out of /var/www/another/whoops.html.

As you can see, understanding the circumstances in which Nginx triggers a new location search can help to predict the behavior you will see when making requests.

Debugging

If you just want to see which block was used, and don't care about returning otherwise valid results, it might be more straight-forward to use return rather than add_header.

```
location / {
    return 200 'location 1';
}

location ~ /(\w+\-)\.html {
    return 200 'location 2';
}

location @named {
    return 200 'location named';
}
```

Examples:

```
location ~ ^/(?:index)\.php(?:$|/)
location ~ ^\/(?:core\/img\/background.png|core\/img\/favicon.ico)(?:$|\/)
location ~ ^/(?:index|core/ajax/update|ocs/v[12]|status|updater/.+|oc[ms]-
```

```
provider/.+)\.php(?:$|/)
```

Test Links

Regex Location tester