16

Dynamic modules: how it works

Maxim Dounin

Basics

Why dynamic modules were introduced in nginx, what problems they solve, how to use them

Basics

- Static modules exist in nginx from the very first version
- Dynamic modules were introduced in nginx 1.9.11, Feb 2016
- Joint work of Ruslan Ermilov and me, Maxim Dounin

Are dynamic modules needed?

- Modules exist for ages and work fine
- Binary upgrade allows changing nginx binaries without dropping requests

That is: you can install nginx with a different set of modules and upgrade.

Problem: hard to maintain packages, especially due to modules with external dependencies.

Main goals

- Simplify package maintenance
- Simplify debugging

Mini how-to

compile:

```
./configure --with-http_image_filter_module=dynamic

./configure --add-dynamic-module=/path/to/module
```

nginx.conf:

```
load_module modules/ngx_http_image_filter_module.so;
```

Implementation

Some details about implementation of dynamic modules

Should be simple

- Lots of 3rd party modules
- Important to minimize required changes in modules, make it easy for module authors to support dynamic loading

Initial trivial solution

Just loading modules from a fixed directory on nginx startup.

- No changes in the code of modules
- Changes to "`config`" scripts still needed

So we implemented something a bit more complex with the `load_module` directive in configuration. This allows to load and unload modules via nginx configuration.

Subtle parts

- Module “`config`” scripts
- List of modules, `ngx_modules`
- Module indexes
- Order of modules
- Signature and version checking
- Filter chains

Module “config” scripts

- Many existing modules, compatibility is important
- Underlying configure code remains mostly the same and uses the same variables
- Introduced `auto/module` script to assign these variables for static compilation
- And it allows compiling modules dynamically as well
- The script is also used by nginx own modules, see `auto/modules`

List of modules: ngx_modules

- List of modules is global, `ngx_modules`
- But we need each configuration to use its own list of modules
- `ngx_modules` list replaced with `cycle->modules`

Module indexes

Each module is described by an `ngx_module_t` structure, and these structures are global. Hence:

- indexes must not be changed
- no conflicts with previous configuration

Loading of modules takes care of this. And `ngx_count_modules()` does this for `ctx_index`'es of sub-modules.

Order of modules

- Originally specified in configure
- Important in some cases, especially for filters: gzip should be before write filter
- No configure-imposed order for dynamic modules
- Each dynamic module remembers desired order

Signature checking

- Structures depend on compilation options
- Expect segfaults when loading a module compiled with different options
- Signature checking to the rescue
- Implemented in existing `NGX_MODULE_V1` macro, no changes required

Version checking

- Structures change from version to version
- Again, segfaults when loading a module compiled with different version
- Version checking to the rescue
- Implemented in existing `NGX_MODULE_V1` macro, no changes required

Filter chains

- Filter chains are global, initialized during postconfiguration callbacks right after parsing configuration
- Configuration can fail after this, e.g., due to conflicting listening sockets
- Rollback to the old configuration will unload just loaded modules
- Inconsistent state: a module is not loaded, yet registered in a filter chain, respawn of dead worker processes won't work
- Changing this will require lots of changes to modules
- If you have a good solution, please share

What we ended up with

- `auto/module` configure script
- `ngx_modules` global replaced with `cycle->modules`
- `ngx_count_modules()` function to assign context indexes
- `load_module` directive

Only complex modules with their own sub-modules actually need code changes. Everything else needs only config script to be converted.

Conversion

How to convert your module to be a dynamic one

Conversion

- change `config` script to use `auto/module`
- `ngx_count_modules()` in complex modules
- `ngx_modules` to `cycle->modules` in complex modules

ngx_count_modules()

```
@@ -144,14 +144,7 @@ ngx_http_block(ngx_conf_t *cf, ngx_comma

     /* count the number of the http modules and set up their indices */

-    ngx_http_max_module = 0;
-    for (m = 0; ngx_modules[m]; m++) {
-        if (ngx_modules[m]->type != NGX_HTTP_MODULE) {
-            continue;
-        }
-
-        ngx_modules[m]->ctx_index = ngx_http_max_module++;
-    }
+    ngx_http_max_module = ngx_count_modules(cf->cycle, NGX_HTTP_MODULE);
```

(http://hg.nginx.org/nginx/rev/0f203a2af17c)

ngx_modules to cycle->modules

```
-     for (m = 0; ngx_modules[m]; m++) {
-         if (ngx_modules[m]->type != NGX_HTTP_MODULE) {
+     for (m = 0; cf->cycle->modules[m]; m++) {
+         if (cf->cycle->modules[m]->type != NGX_HTTP_MODULE) {
              continue;
          }

-         module = ngx_modules[m]->ctx;
+         module = cf->cycle->modules[m]->ctx;

          if (module->postconfiguration) {
              if (module->postconfiguration(cf) != NGX_OK) {
```

(http://hg.nginx.org/nginx/rev/cf5e822cf470)

Config script

Previous version:

```
ngx_addon_name="ngx_http_delay_module"
HTTP_MODULES="$HTTP_MODULES ngx_http_delay_module"
NGX_ADDON_SRCS="$NGX_ADDON_SRCS $ngx_addon_dir/ngx_http_delay_module.c"
```

New version, with `auto/module` script:

```
ngx_addon_name="ngx_http_delay_module"

ngx_module_type=HTTP
ngx_module_name=ngx_http_delay_module
ngx_module_incs=
ngx_module_deps=
ngx_module_srcs=$ngx_addon_dir/ngx_http_delay_module.c
ngx_module_libs=

. auto/module
```

Compatible config version

You may want to make a module dynamic, but still preserve compatibility with old versions of nginx. Test the `$ngx_module_link` variable to do this:

```
ngx_addon_name="ngx_http_delay_module"

if test -n "$ngx_module_link"; then

    ngx_module_type=HTTP
    ngx_module_name=ngx_http_delay_module
    ngx_module_incs=
    ngx_module_deps=
    ngx_module_srcs=$ngx_addon_dir/ngx_http_delay_module.c
    ngx_module_libs=

    . auto/module

else

    HTTP_MODULES="$HTTP_MODULES ngx_http_delay_module"
    NGX_ADDON_SRCS="$NGX_ADDON_SRCS $ngx_addon_dir/ngx_http_delay_module.c"

fi
```

Parameters of auto/module

- **`ngx_module_type`**: module type, usually `HTTP` or `HTTP_AUX_FILTER`
- **`ngx_module_name`**: name of the module
- **`ngx_module_incs`**: include directories
- **`ngx_module_deps`**: dependencies to check when building
- **`ngx_module_srcs`**: source files to build
- **`ngx_module_libs`**: libraries to link with
  - special values: `PCRE`, `OPENSSL`, `ZLIB`, `LIBXSLT`, `LIBGD`, `PERL`, `GEOIP`;
  - or just a lib, **--**`lstdc++`
- **`ngx_module_link`**: linking type, should not be set
  - set automatically to `ADDON` or `DYNAMIC`
- **`ngx_module_order`**: module order to be used, normally set based on module type

Complex modules

It is possible to make a single *.so file with multiple modules in it, just list multiple modules in `ngx_module_name`. The *.so file will be named after the first one.

```
ngx_addon_name="ngx_http_foo_module"

ngx_module_type=HTTP
ngx_module_name="ngx_http_foo_module ngx_http_bar_module"
ngx_module_incs=
ngx_module_deps=
ngx_module_srcs="$ngx_addon_dir/ngx_http_foo_module.c \
                 $ngx_addon_dir/ngx_http_bar_module.c"
ngx_module_libs=

. auto/module
```

All listed modules will be compiled into a single *.so file when using `--add-dynamic-module`, and into nginx itself when using `--add-module`.

How loading works

Module loading explained, and some results of how it works

load_module

- `load_module` calls `dlopen()`
- `dlclose()` in cycle pool cleanup unloads the module

Configuration reload

- `load_module` calls `dlopen()`
- previous configuration freed, calls `dlclose()`
- module remains always loaded

How to upgrade a module?

To upgrade dynamic module on the fly, you can use the same procedure as to upgrade nginx itself:

```
kill -USR2 `cat nginx.pid`
sleep 2
kill -QUIT `cat nginx.pid.old`
```

On most systems

```
service nginx upgrade
```

will do the same.

Further improvements

Things we are working on

Compatibility between builds

- Structures depend on compilation options
- And this complicates building of dynamic modules

We are working on making things easier. Something like:

```
./configure --with-compat
```

will build nginx with various fields always present, regardless of other configure options.

Compatibility with nginx-plus

- Right now it is not possible to load a module compiled for nginx into nginx-plus
- It will be possible as part of the compatibility work

Thank you!

Questions?

Maxim Dounin

mdounin@mdounin.ru