





# What's an Apache Module?

*Sleep through this slide if you already know the answer*

- A way to extend the Web server's request processing
- Needs to be built in an Apache development environment
- Can be statically or dynamically loaded



•  
•  
•

# Why a Module?

- Modules are a very.. well, modular.. way of adding functionality
- Can be a sledgehammer to drive a nail
- Before writing one, figure out what specific function it's supposed to accomplish
- Categorise your module: security, URL manipulation, logging, or content provision

•  
•  
•

# Examples of Apache Modules

- `mod_auth` -- Simple text-based user/password access control
- `mod_speling` -- Real-time correction of simple URL typos
- `mod_rewrite` -- Powerful way to map requests to Web resources
- `<http://modules.apache.org/>`

- 
- 
- 

# The API

- In one sense, the means by which the module hooks into the server
  - Various structures, such as `module`, `command_rec`, `handler_rec`
- In another sense, the features and functions the server provides to modules so they can do their things
- [<http://dev.apache.org/apidoc/>](http://dev.apache.org/apidoc/)

- 
- 
- 

# Module Involvement

- The server invokes the module at various points during configuration, and at various points during request processing..
- ..but only for the types of activities in which the module has registered an interest
  - *e.g.*, content generation, URL parsing, security checking, directive processing, *etc.*

# Operating Environments

- Server config
  - ‘global’
  - *per-virtual* host
- Directory config
  - Called with a directory path:
    - “”, meaning the server-wide default
    - a real directory
- Request processing

- 
- 
- 

# Configuration Hooks

- Server config records
- Directory config records
- Attached to the appropriate structure
  - Both accessed with the same routine, just identifying a different list pointer
- Merging/inheritance
  - *Very* directive-specific

•  
•  
•

# Server Config Hooks

- Not used very often
- Used for things which aren't affected by request details, such as directory aliasing or server management



## *Per*-directory Config

- Used extensively to control behaviour relative to document location
- Examples include
  - labeling file types as indicating specific contents (*e.g.*, “.html” means HTML)
  - applying access control
  - controlling whether directory listings are allowed



•  
•  
•

# Config Processing Phases

- Init and `child_init` phases
- Both server and *per-directory* config environments have two phases associated with them
  - Config record creation
  - Merging a config record with a parent config
- Module is called only for phases it requests

•  
•  
•

# Accessing Config Records

- **Key routine:** `ap_get_module_config()`
- **Indicate the list on which your record should be located**
  - `r->per_dir_config, r->request_config`
  - `s->module_config, s->lookup_defaults`
- **Use `ap_set_module_config()` to put on `r->request_config` list**

- 
- 
- 

# API Structures

- `server_rec` (usually pointed to by “s”)
- `request_rec` (usually pointed to by “r”)
- `module`
- `cmd_parms`
- `pool` (usually pointed to by “p”)

•  
•  
•

## server\_rec Structure

- Contains things like server's names, logfile pointers, timeout values, *etc.*
- May be for the global server or for a virtual host
- Generally a read-only structure for modules

•  
•  
•

## request\_rec Structure

- **Mostly pointers**
  - server\_rec for the server that received the request, name of the file (if any) to which the request resolves, memory pool, HTTP method involved, request and response header fields, environment variable list, module config record list, *etc.*
- **Modules use as both input and output**

•  
•  
•

## cmd\_parms Structure

- Passed as argument to module directive handlers
- Includes info about the config line being parsed, memory pool in which the handler can allocate structures, what types of directives are permitted, server\_rec for the server environment in which the directive occurred, *etc.*

•  
•  
•

## module Structure

- Main means for a module to let the server know what the module wants to do
- Identifies callback points in the module for the core to invoke as processing advances
- Specifies special-handling names, such as “server-parsed” or “cgi-script”
- Points to list of directives for the module

•  
•  
•

## pool Structure

- Arguably the single most important structure in the API
- Used for memory management instead of malloc/free
- Input to almost every API routine
- Modules can specify actions to be taken automatically on garbage collection

- 
- 
- 

# Request Processing

- Phases:
  - Post-read, URI translation, header field parsing, access, authentication, authorisation, MIME type checking/setting, last-minute fixups, content provision, logging
- All phases stop invoking modules immediately if an error is returned
- Some end with first ‘OK’, some call all

- 
- 
- 

# Module Order

- Modules are invoked in the same order for each phase
- Call order is determined by order in the `src/Configuration` file (static load) or the `AddModule` list (dynamic load)
- Order is LIFO; most important/highest priority modules should be listed *last*

- 
- 
- 

## Module Order *(continued)*

- Multi-function modules (*i.e.*, those that hook into multiple phases) *still* get called in the same order, even if it's not appropriate
- Hooks are called once at most *per* request, and can affect the processing of later modules in the same and subsequent phases

- 
- 
- 

# Hook Return Values

- Return values indicate success or failure
  - Directive handlers: NULL, DECLINE\_CMD, or error string
  - Config record handlers: pointer to structure, or NULL
  - Other handlers: OK, DECLINED, or HTTP status number

- 
- 
- 

## Post-read Phase

- Module contributions to this phase are called after the request header has been read
- Used to make decisions or notes about aspects of the request header (such as setting environment variables)
- Examples
  - `mod_setenvif`

- 
- 
- 

# URI Translation Phase

- Used to turn a URI into a filename, or possibly another URI
- Examples
  - mod\_rewrite
  - mod\_alias

- 
- 
- 

# Header Field Parsing

- Called after the request header has been read from the client and the post-read phase has been completed
- Typically not used; purpose was to make decisions or notes about aspects of the header, which are now done in the post-read phase

•  
•  
•

## Security: Access

- Name for the mandatory access control phase, which doesn't depend upon any user-supplied credentials
- Typical usage is allowing/denying access based on IP address of client

# Security: Authentication

- Verifying that the credentials are valid
  - *i.e.*, is the username in the database? Does the password match?
- Called ‘discretionary access control’ since the choice of credentials supplied is at the user’s discretion, and can be varied by it
- Only called if `Require` directive is in scope!

•  
•  
•

# Security: Authorisation

- Occurs after access and authentication checking
- Verifies that the authenticated credentials are allowed to access the resource
- Possible for someone to have a valid username and password from the database, but not the right one for *this* resource

•  
•  
•

# Type Checking/Setting

- Potentially sets the `r->content_type` field according to the IMT of the resource
- Examples
  - `mod_mime`
  - `mod_mime_magic`
  - `mod_rewrite`

- 
- 
- 

# Fixups

- Last chance to make changes to the `request_rec` structure before the server starts sending content to the client
- Examples
  - `mod_env`
  - `mod_speling`
  - `mod_rewrite`

•  
•  
•

# Content Handling

- Surprisingly few modules actually get involved in providing content
  - Examples: `mod_cgi`, `mod_include`, `mod_perl`, `mod_php3`, `mod_info`, `mod_status`
- Content handling is typically more complex than other phases
- Can be either generation (*e.g.*, `mod_status`) or transformation (*e.g.*, `mod_php3`)

- 
- 
- 

# Logging

- Logging phase called after response has been sent to client; failure at this phase doesn't interfere with client's request
- Examples
  - mod\_log\_config
  - mod\_log\_referer
  - mod\_log\_agent

•  
•  
•

# Notes about Module Processing

- The `request_rec` is the *only* currently-supported way of passing information from phase to phase
- Cannot assume the same child will handle two consecutive requests from the same client
- *Always* be modular, thread-safe, and register cleanups

- 
- 
- 

## Other Sources

- Places to look in particular:
  - `<http://dev.apache.org/apidoc/>`
  - `src/modules/example/mod_example.c`
  - `<http://modules.apache.org/>`
- Most importantly, look for modules that do something similar, and see how they do it
  - Modules in the base package are probably better references than 3P modules

•  
•  
•

# Conclusion

- Have a clear idea of what you're trying to accomplish
- Fit your goals into the phase structure
- Use an existing module with similar functionality -- modify it
- Ask questions on USENET
- Join the apache-modules and new-httpd mailing lists