

ENTERPRISE DEVELOPMENT WITH MACH-II: MODULES, PLUGINS, AND FILTERS

#FR3B, Frameworks A - Z

Matt Woodward

Principal IT Specialist, United States Senate

www.mattwoodward.com/blog



CF.Objective()

What We'll Cover

- Mach-II's Modular Architecture
 - a little understanding of how Mach-II works goes a long way
- Event Filters, Plugins, and Modules
 - what they are
 - the differences between them
 - when to use them
 - when not to use them
 - what problems they can help you solve

What We Won't Cover

- Mach-II Basics
- OOP Basics
- I'm happy to answer questions on some of these sorts of things in the context of the presentation, but if you don't have a pretty firm grasp of OO and Mach-II basics, this may not be the talk for you

Who Am I?

- I'll ignore the deeper philosophical implications and give you the basics ...
- Principal IT Specialist, Office of the Sergeant at Arms, United States Senate
- ColdFusion Developer since 1997
 - Also dabble in Flex and Java
- Mach-II Release Coordinator
- Manager, Capitol Hill User Group
- Member of the Open BlueDragon Steering Committee
- Adobe Community Expert for ColdFusion
- Co-Host, ColdFusion Weekly Podcast

Agenda

- Mach-II Architecture
 - how Mach-II is extended at an architectural level
 - how Mach-II loads framework CFCs
 - the purpose of the `configure()` method
 - other examples of extending the framework in this same way (Property CFC, logging, caching, etc.)

Agenda (cont.)

- Filters, Plugins, and Modules: What's the Difference?
- Filters and Plugins: Which to Use When
- Filters and Plugins in Detail
- Modules
 - segment application architecture into larger reusable sub-applications
 - drop existing functionality into applications

Mach-II's Modular Architecture

Extensible By Design



CF.Objective()

Extending Mach-II

- Event Filters and Plugins have been in Mach-II since the early days
- More recently added custom Properties, Modules, more to come (go to Peter's session!)
- All Mach-II objects extend the framework's `BaseComponent.cfc`
 - `configure()` method called on each CFC as the framework loads
 - easy and clean to add new functionality to the framework

Your Custom CFCs

- When using any of these objects in your applications with custom CFCs, you extend the native Mach-II framework CFC
 - this automatically puts your custom functionality into Mach-II's load process, gives you access to Mach-II BaseComponent functionality, etc.

Event Filters, Plugins and Modules: What's the Difference?

Deciding Which to Use When

CF

CF.Objective()

Why Bother With Any Of This?

- Easily layer functionality on top of your application without having any impact on the business logic
 - abstraction, separation of concerns
 - some aspect of not mixing cross-cutting concerns into business logic
- Easily add/remove functionality as needed
- Create modular, reusable functionality
 - can drop these objects into any Mach-II app

Event Filters: The Basics

- As the name implies, event filters (or just “filters”) are used to filter events
- Typically used to conditionally abort the processing of a particular event, or take action on data associated with a particular event or application state
- Contain a single `filterEvent()` function
- Flexible placement within the event-handler

Use a Filter When ...

- You need something to occur only on specific events
- You need simple pass/fail checks on whether or not the event should continue processing
- You need flexibility in terms of placement of the add-on functionality within the event handler

Plugins: The Basics

- Unlike filters, plugins are called automatically on every event
- Plugins provide granular access to various plugin points that occur during the course of the event
- No flexible placement as with filters
- Can programmatically control whether or not specific plugin points execute on specific events

Use a Plugin When ...

- You need something to occur on (more or less) every event
 - the frequency of occurrence is often the decision-making point between an event filter and a plugin, depending on the specific functionality involved
- You need access to various points during the event's execution

Modules: The Basics

- Larger scale modularity than filters and plugins
- Used to modularize entire applications or logical segments of functionality/sub-applications
- Can be used to drop entire pre-existing applications into your Mach-II application
 - this may (usually does) require a very small amount of rework to the existing application

Use a Module When ...

- You want to modularize large sections of your application into reusable, isolated components
 - these sections do not necessarily need to be 100% self-contained
- You want to drop a pre-existing Mach-II application into your application
 - e.g. MachBlog dropped into mach-ii.com as the site's blogging application

Event Filters

Per-Event Functionality and Flow Control



CF.Objective()

Event Filters in Detail

- Filters are declared in the `<event-filters>` section of the `mach-ii.xml` config file
- The `<filter>` command must be included in the body of the event-handler in which you want the filter to be run, at the point at which you want it to run
- Filters contain a `filterEvent()` method that is called when the `<filter>` command is executed

Event Filters in Detail (cont.)

- `filterEvent()` takes three arguments
 - `event`
 - `eventContext`
 - `paramArgs`
- Default parameters may be set for the filter in the `<event-filters>` section of the XML config file, but these may also be overridden within each `<filter>` command

Event Filters in Action

- Checking a user's role to see if they are allowed to see specific data
- Example of default paramArgs that are overridden at the individual <filter> command level

Plugins

Granular Access to Execution
Points



CF.Objective()

Plugins in Detail

- Plugins are declared in the `<plugins>` section of the XML config file
- Unlike filters, plugins are executed automatically on every event
 - there is no `<plugin>` command that corresponds to the `<filter>` command
- Granular access to various points of execution within the event

Seven Plugin Points

- `preProcess()`
- `preEvent()`
- `preView()`
- `postView()`
- `postEvent()`
- `postProcess()`
- `handleException()`

What's Available at Plugin Points?

- The event object may or may not be available at a given plugin point
- `preProcess()`: called at the start of each request
 - has access to `eventContext`
 - current event is in queue but is not returned by `getCurrentEvent()`
- `preEvent()`: called immediately prior to the execution of the `<event-handler>`
 - `eventContext` now contains the current event

What's Available at Plugin Points?

- `preView()`: called immediately prior to the execution of each `<view-page>` command
- `postView()`: called immediately after the execution of each `<view-page>` command
- `postEvent()`: called immediately after the execution of `<event-handler>` command
 - `eventContext` still contains the current event
- `postProcess()`: called at the end of each request
 - `eventContext` no longer contains an event at this point

Plugins and Exceptions

- `handleException()` method is called when an exception occurs
- Be aware that the `eventContext` may or may not contain an event depending on where the exception occurs, the state of the event queue, etc.
- `handleException()` is also passed a `MachII.util.Exception` object

Plugins in Action

- Check login status based on specific event names and event name prefixes
- Role check similar to filter example
 - meant to show overlap between filters and plugins depending on how they're used
- Perform logging at the end of every request
 - new Mach-II 1.6 logging functionality is another option!

Modules

Modular Architecture for Large-Scale Applications



CF.Objective()

Modules in Detail

- Used to create highly modular Mach-II applications
- Used to incorporate existing functionality into Mach-II applications
- Each module within the application lives in its own space
 - event names between modules will not conflict
 - modules have their own XML config files

Modules in Detail (cont.)

- `announceEventInModule()` method handles the announcing of events in specific modules
- `<announce>` command has optional “module” attribute
- `<event-mapping>` command has optional `mappingModule` attribute

Modules: Caveats and Features

- Using existing code that was not designed to be used as a module may require some modifications
- Modules may not have their own modules
 - think one parent module with n number of children
 - each child is at the same level as every other child (all siblings)

Modules: Caveats and Features

- Events may be overridden between modules
 - e.g. can configure a module to call another module's event when an event is announced
- Event syntax uses a colon (:) as the separator between the module name and the event name
 - <http://server/index.cfm?event=MODULE:EVENT>
 - the : separator is user-configurable

Modules in Action

- Dropping a pre-existing authentication module into a Mach-II application

Recap and Q&A

CF

CF.Objective()

Recap

- Use event filters when...
 - you want per-event functionality
 - your needs are pretty simple
 - you need precise placement of the filter
- Use plugins when ...
 - you want functionality executed on every event
 - you need access to specific plugin points

Recap (cont.)

- Filters and Plugins *do* overlap!
 - depending on how they're used and your specific needs, you might be able to use either, but one will likely make more sense than the other
- Use modules when ...
 - you have large chunks of functionality you want to modularize
 - you're building an application with many logical sub-applications
 - you want to use pre-existing functionality/applications within your application

Questions?

- Thanks!
- Matt Woodward
mpwoodward@gmail.com
<http://www.mattwoodward.com/blog>
<http://www.coldfusionweekly.com>