Mach-II Application Configuration

Mach-II 1.1.1 for ColdFusion

Last Updated on November 15, 2006

Benjamin Edwards
Mach-II Framework

Overview

Configuring a Mach-II application involves editing two files: mach-ii.xml and index.cfm. Use the Mach-II Application Skeleton download from mach-ii.com to make creating and configuring Mach-II applications easiest.

For each sample application provided on www.mach-ii.com the index.cfm file is in the application's root folder (i.e. ContactManager/index.cfm) and the mach-ii.xml file is in a sub-directory called config (i.e. ContactManager/config/mach-ii.xml).

The Mach-II framework code is a toolkit (code library) that can be installed once and used from the same location for each Mach-II application. Currently the framework files (the MachII directory and everything in it) need to be installed in the wwwroot of the ColdFusion server, or equivalently mapped. Absolute CFC paths (i.e. MachII.framework.AppManager) are used for referring to components.

index.cfm (or Application.cfc extending MachII.mach-ii)

The index.cfm file is typically the entry point for a request into the Mach-II framework. There are several important parameters for the application and framework that should be set in index.cfm (or equivalent file, Application.cfc, wich extends MachII.mach-ii). They are:

- MACHII_CONFIG_PATH
- MACHII_CONFIG_MODE
- MACHII_APP_KEY
- MACHII_VALIDATE_XML
- MACHII_DTD_PATH

The latter two are new with Mach-II version 1.1.0 and are optional.

MACHII_CONFIG_PATH
This variable needs to be set to the absolute path of your application's configuration file (usually named mach-ii.xml). This can be accomplished in ColdFusion by using the built-in ExpandPath() function passing a relative path to the mach-ii.xml file.

MACHII_CONFIG_MODE
This variable needs to be set to a value of –1, 0, or 1. These number values represent when the configuration file (mach-ii.xml) will be reloaded to reconfigure the application's framework components.

- **-1 (never reload):**
  do not reload the configuration file (recommended for production deployments)
- **0 (dynamic reload):**
  reload the configuration file whenever the mach-ii.xml file is updated by

automatically checking the file's date last modified (recommended for development)

- **1 (always reload):**
  reload the configuration for each request (will result in significant performance penalties and is not recommended for production)

Each time the configuration is reloaded the application framework is reinstantiated. This means lots of object instances are created and the mach-ii.xml file is reread from file and reparsed. This also means any state information held in framework components (such as Listeners) will be reset if not persisted.

MACHII_APP_KEY
This variable needs to be set to a value that is unique amongst any Mach-II subapplications that are defined for the same cfapplication. This property enables multiple Mach-II applications, each independent of each other, to operate under one cfapplication. By default the app-key value is set to the name of the folder in which the application is stored.

Example code from an index.cfm file:

```
<!--- Set the path to the application's mach-ii.xml file. --->
<cfparam name="MACHII_CONFIG_PATH" type="string" default="#ExpandPath('./config/mach-ii.xml')#" />
<!--- Set the configuration mode (when to reload). --->
<cfparam name="MACHII_CONFIG_MODE" type="numeric" default="0" />
<!--- Set the app key for sub-apps within a single cf-application. --->
<cfparam name="MACHII_APP_KEY" type="string" default="#GetFileFromPath(ExpandPath('.'))#" />
<!--- Whether or not to validate the configuration XML before parsing. --->
<cfparam name="MACHII_VALIDATE_XML" type="boolean" default="false" />
<!--- Set the path to the Mach-II's DTD file. --->
<cfparam name="MACHII_DTD_PATH" type="string" default="#ExpandPath('/MachII/mach-ii_1_1.dtd')#" />
```

mach-ii.xml

The mach-ii.xml file is the main configuration file for a Mach-II application. The XML configuration file has an accompanying DTD currently called mach-ii_1_1_1.dtd that is included with the framework code files when downloaded.

<mach-ii> : [version]
The root element of a Mach-II configuration file is a <mach-ii> element with an optional version attribute. The version attribute should specify the version of Mach-II the configuration file is supposed to use.

The Mach-II XML configuration file (config file) has six main elements for defining a Mach-II application and its components.

Example code from a mach-ii.xml file:

```
<mach-ii version="1.1">
    <!-- PROPERTIES -->
    <properties>
        ...
    </properties>
    <!-- LISTENERS -->
    <listeners>
        ...
    </listeners>
    <!-- EVENT-FILTERS -->
    <event-filters>
        ...
    </event-filters>
    <!-- PLUGINS -->
    <plugins>
        ...
    </plugins>
    <!-- EVENT-HANDLERS -->
    <event-handlers>
        ...
    </event-handlers>
    <!-- PAGE-VIEWS -->
    <page-views>
        ...
    </page-views>
</mach-ii>
```

<properties>
<property> : name, value

The <properties> element allows you to configure the framework instance for an application. Six properties need to be defined in each config file for the framework to operate properly. These properties are each defined as a <property> element under the <properties> of the config file.

The required properties to be defined:

- **applicationRoot**
  The path to the application's folder relative to the web-server root (i.e. '/SampleApp')
- **defaultEvent**
  The event for the framework to handle if one is not specified in a new request
- **exceptionEvent**
  The event for the framework to handle if there's an unhandled exception in the application
- **maxEvents**
  The maximum number of events for the framework to process on a single request
- **eventParameter**
  the name of the request parameter (form or url) that will define the event for the framework to handle
- **parameterPrecedence**
  form | url, which to favor for conflicting request parameters

Example code from a mach-ii.xml file:

```
<mach-ii>
    <!-- PROPERTIES -->
    <properties>
        <property name="defaultEvent" value="showHome" />
        <property name="exceptionEvent" value="exception" />
        <property name="applicationRoot" value="/ContactManager" />
        <property name="eventParameter" value="event" />
        <property name="parameterPrecedence" value="form" />
        <property name="maxEvents" value="10" />
    </properties>
    . . .
</mach-ii>
```

<listeners>

<listener> : name, type

The <listeners> element allows you to register listeners for an application. The name attribute specified for a <listener> element should be unique amongst listeners. The type attribute specified should be a full path to a CFC.

<invoker> : type

Also for each listener an invoker may be specified with a child <invoker> element. The type attribute for the <invoker> element should be a full path to a CFC. Mach-II Listeners can also have a <parameters> and <parameter> elements specified to define additional configuration information.

Example code from a mach-ii.xml file:

```
<mach-ii>
...
    <!-- LISTENERS -->
    <listeners>
        <listener name="ContactManager" type="ContactManager.model.ContactManager">
            <invoker type="MachII.framework.invokers.EventArgsInvoker" />
            <parameters>
                <parameter name="param1" value="value1" />
            </parameters>
        </listener>
    </listeners>
...
</mach-ii>
```

<event-filters>

<event-filter> : name, type

The <event-filters> element allows you to register event filters for an application. The name attribute specified for an <event-filter> element should be unique amongst event filters. The type attribute specified should be a full path to a CFC.

Event-Filters can also have a <parameters> and <parameter> elements specified to define additional configuration information.

Example code from a mach-ii.xml file:

```
<mach-ii>
...
    <!-- EVENT-FILTERS -->
    <event-filters>
        <event-filter name="simpleFilter" type="Test.filters.SimpleFilter " />
            <parameters>
                <parameter name="param1" value="value1" />
            </parameters>
        </event-filter>
    </event-filters>
...
</mach-ii>
```

<plugins>

<plugin> : name, type

The <plugin> element allows you to register plugins for an application. The name attribute specified for a <plugin> element should be unique amongst plugins. The type attribute specified should be a full path to a CFC.

Plugins can also have a <parameters> and <parameter> elements specified to define additional configuration information.

Example code from a mach-ii.xml:

```
<mach-ii>
...
    <!-- PLUGINS -->
    <plugins>
        <plugin name="simplePlugin" type="Test.plugins.SimplePlugin">
            <parameters>
                <parameter name="param1" value="value1" />
            </parameters>
        </plugin>
    </plugins>
...
</mach-ii>
```

<event-handlers>

<event-handler> : event, [access]

The <event-handlers> element allows you to define how to handle an application's events. The event attribute specified for an <event-handler> element should be a unique event name. The access attribute specified for an <event-handler> element is optional, and if defined, must have a value of public or private. Event-handlers are assumed to be public by default.

The access attribute specified for an <event-handler> element determines when an event can be announced. Events specified in a request parameter must have access specified as public. Events announced via listeners in the same application can be public or private.

Example code from a mach-ii.xml file:

```
<mach-ii>
...
    <!-- EVENT-HANDLERS -->
    <event-handlers>
        <event-handler event="showHome" access="public">
            ...
        </event-handler>
    </event-handlers>
...
</mach-ii>
```

Each <event-handler> can specify any of the following elements in its body.

<announce> : event, [copyEventArgs]
An <announce> element will announce a new event. The copyEventArgs attribute specifies whether or not to copy the event-args of the current event being handled.

Example code from a mach-ii.xm l file:

```
<event-handler event="editUser" access="public">
    <announce event="showEditUserForm" copyEventArgs="true" />
</event-handler>
```

<event-arg> : name, [value], [variable]
An <event-arg> element will set an argument in the current event being handled. The name attribute is the name of the arg to create in the event. The optional value attribute may specify a literal value for the arg. The optional variable attribute may specify an in scope variable to set as the value of the arg. Either the value or variable attribute should be set in an <event-arg> element, not both. If the variable specified is not defined a blank string is set as the value.

Example code from a mach-ii.xml file:

```
<event-handler event="editUser" access="public">
    <event-arg name="submitFormEvent" value="editUser" />
    <event-arg name="user" variable="session.user" />
    ...
</event-handler>
```

<event-mapping> : event, mapping
An <event- mapping> element will set a temporary mapping in the framework that will map one event (if announced) to another. The mapping exists only for the life of the current event being handled.

Example code from a mach-ii.xml file:

```
<event-handler event="createContact" access="public">
    <event-mapping event="contactAdded" mapping="showContact" />
    ...
</event-handler>
```

<event-bean> : name, type, [fields]
An <event-bean> element will create a bean inside the current event. The name of the bean is specified with the name attribute. The type of the bean to create is a fully qualified CFC path specified by the type attribute. The optional fields attribute specifies which fields of the event to use to call setters on the bean.

Example code from a mach-ii.xml file:

```
<event-handler event="createContact" access="public">
    <event-bean name="address" type="model.Address" fields="street,city,state,zip" />
    ...
</event-handler>
```

<filter> : name
A <filter> element allows you apply a filter to the event being handled. The name attribute needs to reference a specified <event-filter> name. One or more <parameter> elements may also be specified for a filter.

Example code from a mach-ii.xml file:

```
<event-handler event="editContact" access="public">
    <filter name="loggedInFilter">
        <parameter name="invalidEvent" value="showLoginForm" />
    </filter>
</event-handler>
```

<notify> : listener, method, [resultArg], [resultKey]
A <notify> element will invoke a listener's method using the current event and its arguments. The listener attribute is the name of a registered listener. The method attributes is a method of the listener to invoke. The optional resultArg argument is an event-arg name to store the returned result of the method call in. The optional resultKey argument is a variable name to store the returned result of the method call in. The resultArg and resultKey attributes may both be used at the same time.

Example code from a mach-ii.xml file:

```
<event-handler event="createContact" access="public">
    <notify listener="contenListener" method="create" resultArg="newContact" />
</event-handler>
```

<view-page> : name, [contentArg], [append], [contentKey]
A <view-page> element will invoke a page-view and optionally store the generated content instead of outputting it. The name attribute is that of a registered pageview. The optional contentArg argument, if present, stores the generated page-view content in the specified event-arg instead of outputting. The optional contentKey argument, if present, stores the generated page-view content in the specified variable instead of outputting. The optional append argument, defaults to false, and determines whether the contentArg/content Key value is appended to (or overwrites) the old value. The contentArg and contentKey attributes may both be used at the same time.

Example code from a mach-ii.xml file:

```
<event-handler event="showContactForm" access="public">
    <view-page name=”header” contentArg=”header” append=”true” />
    <view-page name="contactForm" />
</event-handler>
```

<redirect> : event, [url], [args], [eventParam]
A <redirect> element will redirect (using cflocate) to another event. The event attribute specifies what event to redirect to. The optional url attribute specifies what url to relocate to (defaults to index.cfm). The optional args attribute is a comma-delimited list of the event-args to copy to the redirect url. The optional eventParam attribute specifies the event-parameter in the url. The full url of the redirect will be url?eventParam=event&args.

Example code from a mach-ii.xml file:

```
<event-handler event="cartCheckout" access="public">
    <redirect event=”cartReceipt” />
</event-handler>
```

<page-views>
    <page-view> : name, page
The <page-views> element allows you to register page-views for an application. The name attribute specified for a <page-view> element should be unique amongst page-views. The page attribute specified should be a path to a CFM page relative to the application’s root.

Example code from a mach-ii.xml file:

```
<mach-ii>
...
    <!-- PAGE-VIEWS -->
    <page-views>
        <page-view name="mainTemplate" page="/views/mainTemplate.cfm" />
    </page-views>
...
</mach-ii>
```

About The Author:

Benjamin Edwards is a Sun Certified Java Programmer and holds a degree in Computer Science from the Georgia Institute of Technology with specializations in Software Engineering and Educational Technology. Ben co-founded Synthis Corporation while at Georgia Tech. Ben currently trains developers on software engineering practices with a focus on Java and Object-Oriented programming.

You can contact Ben at ben@mach-ii.com.

Copyright: