Building a Robust Error Handler

Posted At : February 11, 2014 11:25 AM | Posted By : Mark Kruger
Related Categories: ColdFusion, Coldfusion Tips and Techniques

If you have been around the ColdFusion world as long as the Muse you have heard of Mary Jo Sminkey. Mary Jo built a popular ColdFusion ecommerce platform called CFWebstore. She has vast experience in ColdFusion and a seemingly boundless fountain of energy. Her eclectic interests range from technology to baking to dog training. As far as CF Webtools and the Muse can tell, Mary Jo excels at everything she does. We frankly suspect she is actually twins or triplets pretending to be only one person :) The following article is by Mary Jo and details her approach to application specific error handling. She has a detailed and thorough knowledge of the topic. Using this approach she has been able to reduce the number of errors on a very high traffic E-commerce site to practically nil. In the first of 2 articles MJ (as we call her with great affection) details the structure and usage of the handler.

Building a Robust Error Handler (by Mary Jo Sminkey)

Let's face it, sometimes we put less effort into the error handler than into the rest of our code. We might put something in place that throws up a "user friendly" page, and maybe email a dump of the catch or error structure, but when the site goes live, and we are deluged with errors due to search bots, hack attempts and poorly coded pages we turn it off or send all those emails to a seldom-visited mailbox. Sometimes we implement error handling as cftry/cfcatch blocks that do little more than preventing errors from being thrown, instead of helping us track down the issue.

I look at the error handler as a way to help make a site as bug-free as possible. By having it email me as much information as possible about errors, I troubleshoot, fix and patch, and get to a point where errors are the exception rather than the rule. In this article, we'll look at building a single-page, comprehensive error handler. In a future article, we'll look at integrating that error handler with the open source bug tracker BugLogHQ. Before we begin with our error handler let's talk about our error handling strategy.

Error Handling Strategy

Obviously global error handling is a last ditch attempt to catch errors, and only part of a complete strategy for trapping errors. This article assumes you are using things like cftry/cfcatch in your code to handle expected issues that can arise (not just to hide errors). For example, on one application with a controller approach (a main logic file that dictates behaviors and actions), the controllers include many cfparams for URL variables. I updated the cfparam code with a cftry/cfcatch block to trap any typical hack attempts that throw invalid data into the URLs. My catch block redirects those requests to an error page specifically for them. We include that error page in our analytics so we can see how often it gets hit and from where, but I certainly don't need to get regular error emails from those kinds of hits.

The global error handler I use is intended to find and handle any ColdFusion errors that were *not anticipated*, although it can be used inside a cfcatch block as well. More about that approach later. The main goal is to present the user with a friendly error page that doesn't expose any sensitive information, and provides them with some additional guidance. Meanwhile it sends the developers information they can use to try and determine what specifically caused the error.

That brings me to the first choice we need to make. Should we use an onError method or the cferror tag? Here's MaryJo's first big tip. If you use cferror with cfcatch you will have access to all the scopes that ColdFusion has to offer. If you use onError( ) many of these scopes will not be included. But onError() is a part of the new "application.cfc" while cferror is a part of the old style Application.cfm right? How do you use cferror and still implement Application.cfc? Here's how I do it.

```
<cffunction name="OnRequestStart" access="public" returntype="boolean" output="false" hint="Fires on every page request.">
<cferror type="exception" template="/errors/errorhandler.cfm">
...
</cffunction>
```

As you can see I put the cferror line into my onRequestStart. This allows me the benefits of the cferror tag without the limitations of onError. Note that you have to use a tag based Application.cfc which can bump some folks.

I also code my error handler so it can be used within a cftry/cfcatch block which can be particularly useful if I am troubleshooting a specific error in a cfcomponent and need to include the local scope. Normally the "local" scope is not picked up by the global error handler so if I need it, I can copy it to request scope inside the cfcatch block and then pass the error to the error handler to send me the results. However there are some differences in how you receive the error information from a global cferror tag versus a cfcatch, so both will need to be checked in the error handler.

Structuring Your Error Handler

Let's look at a few "setup" tasks for our handler that will help us define what it is going to do.

First, how will we deal with bots and crawlers and their tendency to throw really crazy query strings, cookies, etc. at your site? You may decide to let these errors through, particularly when first launching a site. One reason might be as a post launch check - to be sure that you are properly sanitizing data and using cftry/cfcatch blocks where appropriate. Often however, these just create unnecessary error emails and can be excluded and ignored. Of course your global error handler should still ensure that such attacks don't do anything more than end up on an error page. How do you do it? I use the CGI.HTTP_USER_AGENT against a list of strings to figure out if the user is a bot and then I set a variable I can use later when an error is thrown. For example (using a local struct to set all my variables so it will be easy to exclude later):

```
<cfset localVars = structNew() />
<cfset localVars.crawler = false />
<cfset localVars.browserExcludeList = "bot,spider,crawl,jeeves,yahoo,slurp" />
<cfloop index=" localVars.item" list="# localVars.browserExcludeList#">
    <cfif FindNoCase(localVars.item, CGI.HTTP_USER_AGENT)>
        <cfset localVars.crawler = true />
    </cfif>
</cfloop>
```

Then I can check the variable and if it is a bot I can choose to ignore or exclude the error.

Second, I need to determine which error scope I have. Remember I'm using my handler from both the application.cfc page (as an application wide error handler) and as a specific handler from a cftry/cfcatch block. We'll use that to set a variable to track the error that was captured.

```
<cfif isDefined("error.Diagnostics")>
    <cfset localVars.errormess = error.Diagnostics>
    <cfset localVars.errorData = error>
<cfelseif isDefined("cfcatch.message")>
    <cfset localVars.errormess = cfcatch.message>
    <cfset localVars.errorData = cfcatch>
</cfif>
```

Next I list out all the variable scopes I want to include. Some of these will not always be available, particularly those associated with specific CF tags like CFHTTP or CFFILE, but we want to include as much information in our emails as we possibly can when they *are* available. Your list will depend on the CF functions you are running, but this list is pretty comprehensive.

```
<cfset loclaVars.varstodump="CFCATCH,ERROR,APPLICATION,ARGUMENTS,ATTRIBUTES,CALLER,CGI,CLIENT,CFHTTP,FILE,FORM,REQUEST,SERVER,SESSION,COOKIE,THIS,THISTAG,URL,VARIABLES" />
```

Note, you must also consider whether your site will have any sensitive data that should be excluded from your data dumps when sending emails. If you run an ecommerce site for instance, you will want to exclude any credit card data from the emails. If you have user accounts, you should exclude user passwords (and any other private user data like SSNs, birth dates etc.). To make this happen I create a list of the variable names used on the site that can potentially appear in any of our variable, so I can weed them out when I do the output. I usually exclude the CF session cookies (CFID/CFTOKEN/JSESSIONID) as well.

```
<cfset localVars.securevars="CFID,CFTOKEN,JSESSIONID,SessionID,URLToken,username,password ,passwordnew,ccNumber,cvv2" />
```

Finally, we create a list of scopes that we want to search and scrub for this secure data:

```
<cfset localVars.varstoscrub="ATTRIBUTES,ARGUMENTS,CGI,FORM,REQUEST,SESSION,URL,VARIABLES,COOKIE,ERROR,CFCATCH" />
```

Now we are ready to loop through our list of scopes and dump the data. Some of the things to note in this code that may or may not apply to yours:

1. Since I typically store lots of components in Application memory, I do some special handling to the Application scope to *not* include these in the dump. All of my objects have a prefix of "obj" so it's easy to exclude them using that. If you use a bean factory instead, you'll want to exclude that in a similar fashion.
2. For the rest of my scopes, I set a list of variable names that I want to exclude, typically large elements that don't have any useful data. These include "GeneratedContent" which is part of the Error scope and seldom has anything useful for debugging purposes. I also include "CFError", "Error" and "CFCatch" which duplicate information that is part of the error scopes but is found in variables scope. This will help reduce the amount of data being returned in my emails and make it easier to parse through.
3. When dumping the Variables scope I set a list of elements that are irrelevant for my application and exclude them as well. For example, there's no reason to include a UDF in my scope dump.
4. For the scopes I've flagged to scrub I copy the entire scope key by key, replacing any secure data items found with the word "removed".
5. Our application also includes some elements in cookies that are encrypted (but not particularly secure). To make debugging easier, I add an un-encrypted version of any of these that are found into the error dump as well.
6. Sometimes the same cookie shows up more than once. I check to make sure that a key doesn't already exist before creating it when running the code that includes the cookie scope.

Output Your Data

With all that in mind here is the code to create these dumps. I use cfsavecontent to save them into a variable which I will use in my email:

```
<cfsavecontent variable="localVars.dataDump">
<cfloop list="#localVars.varstodump#" index="localVars.loopItem">

    <!--- Don't expand the application memory var with all the components --->
    <cfif IsDefined("#localVars.loopItem#") AND localVars.loopItem IS "APPLICATION">
         <cfset localVars.ApplicationVars = StructNew()>
         <cfloop item="localVars.each" collection="#Application#">
            <cfif Left(localVars.each, 3) IS NOT "obj" >
                <cfset localVars.ApplicationVars[localVars.each] = Application[localVars.each]>
            </cfif>
        </cfloop>
         <cfdump var="#localVars.ApplicationVars#" label="Application">
        <!--- search and remove secure items --->
         <cfelseif IsDefined("#localVars.loopItem#") AND ListFind(localVars.varstoscrub, localVars.loopItem)>
             <cfset localVars.ScrubbedVars = structNew()>
            <cfset localVars.scopeToCheck = Evaluate(localVars.loopItem) />
             <cfloop item="localVars.each" collection="#localVars.scopeToCheck#">
                <cfif ListFindNoCase(localVars.securevars, localVars.each)>
                    <cfset localVars.ScrubbedVars[localVars.each] = "removed">
                <cfelseif localVars.loopItem IS "COOKIE" AND localVars.each IS "CUSTOMERNAME" OR localVars.each IS "CUSTOMERID">
                    <cfset localVars.ScrubbedVars["unencrypted_" & localVars.each] = cookie[localVars.each] />
                    <cftry>
                        <cfset ScrubbedVars[localVars.each] = application.objCrypt.decryptString(cookie[localVars.each]) />
                    <cfcatch type="Any" >
                        <cfset localVars.ScrubbedVars[localVars.each] = "encrypted">
                    </cfcatch>
                    </cftry>
                <cfelse>
<!-- elements we don't need to output including variables from this error handler page --->
    <cfset localVars.listremove = "GeneratedContent,CFCatch,CFError,Error,localVars">
    <cfif NOT ListFindNoCase(localVars.listremove,localVars.each) AND NOT IsCustomFunction(localVars.each) AND NOT structKeyExists(localVars.ScrubbedVars, localVars.each)>
        <cfset localVars.ScrubbedVars[localVars.each]= localVars.scopeToCheck[localVars.each]>
    </cfif>                </cfif>
            </cfloop>
            <cfdump var="# localVars.ScrubbedVars#" label="# localVars.loopitem#">
     <cfelseif IsDefined("#localVars.loopItem#")>
             <cfdump var="#Evaluate(localVars.loopItem)#" label="# localVars.loopItem#">
     </cfif>
    </cfloop>
    </cfsavecontent>
```

I create a short summary from the actual error data of the error to display at the top of emails. This is useful because many times the summary will clue me in on the error without the necessity of digging through all of the data output.

```
<cfsavecontent variable=" localVars.basicinfo">
    <cfoutput>
        <table width="100%" cellpadding="5" cellspacing="0" border="0">
        <cfloop collection="# localVars.errorData#" item="i">
            <cfset data = localVars.errorData[i]>
            <cfif IsSimpleValue(data) AND i IS NOT "GeneratedContent">
            <tr>
                <td><font face="verdana" size="2"><strong>#Ucase(i)#:</strong><br/>#data#</font></td>                    </tr>
            </cfif>
        </cfloop>
    </table>
    </cfoutput>
</cfsavecontent>
```

Now we need to email all of this carefully collected and sanitized data to a developer. This is where things can get hairy. If you have ever used an error handler that sends emails on a high traffic website, you've probably experienced the fun of waking up to a mailbox full of hundreds, even thousands of errors. Sites are hit by a hackers. Databases go down. Dozens of things are outside of your control and can potentially flood you with error information. To make sure that we don't get buried in emails, we want to track which errors we've emailed and only send a specific error once per interval - say every 4 hours. I also want to know if a given error is triggered multiple times, so I'll keep a count of the number of times it's been hit and resend the error every 25 hits. Your approach will depend on the level of traffic and stability of your code.We also use a server cluster so we add additional code that will inform us as to which server encountered the error specifically.

Assuming we have a struct created during onApplicationStart called "ErrorLog", here's our code for sending and tracking the error emails.

```
<cfif NOT StructKeyExists(Application.ErrorLog, localVars.Errormess) OR DateCompare(Application.ErrorLog[localVars.Errormess].TimeFirstError, DateAdd("h", -4, Now() ) ) LT 0 >
        <cfset hostaddress = createObject("java", "java.net.InetAddress").localhost.getHostName() />
        <cfmail to="errors@cfwebtools.com" from="noreply@cfwebtools.com" subject="Error on #cgi.server_name#"
type="HTML">
     <br/><br/>
            An error has been encountered on #hostaddress#:<br/><br/>
            # localVars.basicinfo#<br/><br/>
            <strong>Full message:</strong><br/><br/>
            # localVars.dataDump#<br/><br/>
    </cfmail>
        <!--- Log the error in the application error log --->
        <cfset Application.ErrorLog[localVars.errormess] = StructNew() />
        <cfset Application.ErrorLog[localVars.errormess].TimeFirstError = Now() />
        <cfset Application.ErrorLog[localVars.errormess].TimeLastError = Now() />
```

```
        <cfset Application.ErrorLog[localVars.errormess].Count = 1 />

<cfelse>
        <!--- Increment the counter and the last error time. --->
        <cflock scope="application">
            <cfset Application.ErrorLog[localVars.errormess].TimeLastError = Now() />
            <cfset Application.ErrorLog[localVars.errormess].Count++ />
        </cflock>
        <!--- If error has occurred 25 in the last four hours, send another email --->
        <cfif Application.ErrorLog[localVars.Errormess].Count MOD 25 EQ 0>
            <cfset hostaddress = createObject("java", "java.net.InetAddress").localhost.getHostName() />
            <cfmail to="errors@cfwebtools.com" from="noreply@cfwebtools.com" subject="MULTIPLE errors on #cgi.server_name#"
         type="HTML">
          <br/><br/>
          The following error has occurred #Application.ErrorLog[localVars.errormess].Count# times in the last four hours on #hostaddress#:
                  # localVars.basicinfo#<br/><br/>
                  <strong>Full message:</strong><br/><br/>
                  # localVars.dataDump#<br/><br/>
          </cfmail>
        </cfif>
    </cfif>
```

That's it! Don't forget to cfinclude a file to display to your users with a friendly "Sorry an error was encountered and the webmaster has been notified" kind of message. And unlike many times when we see this as users, the webmaster really HAS been notified! Not only that, you won't just find out you have an error and see information on the file and line number. Instead, you will have access to a ton of information - such as the form data the user submitted, their cookies, local variables etc. You will have a whole array of data to help you debug and solve the problem. By using this error handling approach, we've been able to reduce errors on our high traffic production site to a rarity in spite of thousands of visitors per hour.

Next time we'll look at how we can use our error handler and leverage a great open source tool, BugLogHQ to do even more logging and tracking of our errors.