Building a Robust Error Handler Part 2

Posted At : July 12, 2017 5:03 PM | Posted By : Mark Kruger
Related Categories: ColdFusion, Coldfusion Tips and Techniques

The Muse is fortunate to have a number of very talented and smart developers work for him. Mary Jo Sminkey has been with CF Webtools for several years now and we simply love having her around. She's a fantastic developer and a fantastic resource. You might catch her at **cf.objective()** where she will be speaking next week.

Meanwhile, you might remember part 1 of this series - **Building a Robust Error Handler**. Below is part 2 in that 3 part series. Enjoy!

Well it's taken a couple years to get around to it, but at long last here is part 2 of my error handling post! Of course, in that time I've continued to work on improvements to my error handler, so many changes that I'm going to need to break THIS post up into multiple parts as well. I'll try not to take too long to get the final part out to you!

So in part one we looked at things like using the error handler for both global errors as well as in cfcatch blocks, dumping all the available scopes while scrubbing them for any sensitive data, keeping track of already reported errors to prevent receiving tons of duplicate emails for the same error, etc.

In this post (part 2) we will expand the functionality of the error handler and talk about strategies for use throughout the application. Let's start with how to handle errors "in depth".

While being able to call my error handler from a cfcatch is very useful, sometimes I want to include it at other points in the code that are not in a try/catch block but that I just want to email some additional information about what's happening to track some kind of issue, without actually throwing an error. So I added the ability to do that by setting a request variable called "errorname" and include that in my code before I include my error handler (along with another request variable for the data struct that I want to include as the 'error' object). I've also added a default (cfelse) condition for any calls to the error handler that don't have this value set. As a reminder, I use a structure in variables scope called "localVars" in my error handler that I can easily leave out of the error dumps that are done.

```
<cfif isDefined("error.Diagnostics")>
...
<cfelseif isDefined("request.errorname") >
<cfset localVars.errormess = request.errorname>
<cfset localVars.errorData = structKeyExists(request,"errorData") ? request.errorData : {} >
<cfelse>
<cfset localVars.errormess = "Unknown Error">
<cfset localVars.errorData = {} >
</cfif>
```

I can then drop my error handler anywhere in the code that I want to capture and log information about what is happening at that point in the code, particularly helpful on production sites to track unexpected behavior without throwing an error that will interrupt the user's action.

Now, it's very frustrating when using a global error hander that when you have an error in something like a CFC, the local scope (which includes the arguments) is lost by the time it gets to the error handler. So one way around this is to just make a copy of your local scope before you throw to the error handler. Like this:

```
<cfscript>
function doSomething() {
try {
    all the important stuff here....

} catch (any err) {
    request.localData = duplicate(local);
rethrow;
}
</cfscript>
```

This copies all you local scoped data into the request scope and then rethrows the error so it will go to the error handler. You can of course add any other error handling you might need specific to your CFC methods in here as well but this ensures that you aren't missing crucial data in the error handler output. Don't forget to include any variables that could potentially have secure data in the sanitize list in the error handler! I routinely review all errors coming through to make sure I didn't overlook something that might not be appropriate to include in an email (passwords, credit card, banking information, etc.)

Now, I may want to show a user-friendly error message when the error handler is called, but a lot of times I may want to just continue on after logging and/or emailing the error (such as in the above case where I am just capturing data and emailing it to diagnose an issue but don't want to interrupt the process). So I added another request variable called "errorType" which I can set prior to calling the error handler which determines the action to take at the end of the process. I include cfparams for this in the above block so that I have a default setting for each type of error as well. So I have a value of "fatal" for global errors, "error" for a caught (cfcatch) error, "custom" for my custom generated errors, and "unknown" for anything else. Some common uses I have in my error handler for other error types are "email" for errors I just want to send an email for and then continue, and "startup" errors when my initialization process is not working and I need to display a custom error message due to application variables, wirebox, etc. not being available.

The next update I've made to error handling is more an application-wide logging addition I've made. Whenever looking at my error messages, I would be frustrated at how often the scope data alone is not enough to figure out what happened. I wanted to know not only what the user was doing when the error occurred... but what they did that led up to the error. So I added two elements to the session data, a page tracker that keeps a list of the last 40 pages the user requested. And an Ajax tracker that keeps a list of the last 40 Ajax calls that were made (since we use Ajax extensively on the site) including the data sent in the request and in some cases, the response as well, or whether the request returned a success or not. We'll cover the Ajax logging in part three of this error handling series, today we'll just tackle the page tracker. Keep in mind that this can use significant memory so if you have a busy site in particularly make sure you have plenty of resources to handle this, and don't set your session timeouts too long such that they are hanging around a lot longer than needed (we do check for search bots and set the session timeout for them to 1 sec so that their sessions time out right away).

Step one is to initialize our page tracker. We'll use an array so we just need to add this to the onSessionStart() method in Application.cfc:

```
session.pageTracker = [];
```

I then added this method to our CFC that is used for handling session management. Our site uses a homegrown MVC with all requests going through "controller.cfm" and "action" is the variable determining which section of the controller to run. Any MVC style application can use a similar method to log all page requests if it doesn't already have something built in, but even non-MVC sites can integrate something similar into their onRequestStart in Application.cfc. We exclude both Ajax calls (path has 'remote' in it) as well as some controller actions that we don't want to include like 404 errors.

```
<cfscript>
public void function logPageRequest( numeric maxPages=40 ) {
if ( NOT findNoCase("/remote",cgi.script_name) ) {
//list of actions to not include in pageTracker
var excludeList = "JSerror,404error";
var thisPage = cgi.script_name & '?' & cgi.query_string;
var pageAction = structKeyExists(URL,"action") ? URL.action : '';
if (!ListFindNoCase(excludeList, pageAction ) ) {
    // skip duplicate page requests
    if ( arrayLen(session.pageTracker) IS 0 OR session.pageTracker[1] IS NOT thisPage ) {
     lock scope="session" type="exclusive" timeout="10" {
     if ( arrayLen(session.pageTracker) LT arguments.maxPages ) {
        arrayPrepend(session.pageTracker, thisPage);
     } else {
        arrayDeleteAt(session.pageTracker, arrayLen(session.pageTracker));
         arrayPrepend(session.pageTracker, thisPage);
        }
     }
     }
}
}
}
</cfscript>
```

Now all we have to do is drop our tracker into the onRequestStart() method in Application.cfc. I use wirebox to handle the CFCs and dependency injection so it looks like this:

```
<cfscript>
request.sessionManager = application.wirebox.getInstance('sessionManager');
request.sessionManager.logPageRequest();
</cfscript>
```

If you want to add the ability to log site actions like stepping through the checkout process, when it doesn't involve page requests (something you may commonly need to do for AngularJS sites for instance), or for things like button clicks, form submissions, etc. you can create a logging method for "actions" as well. Here's the code I use for that. I'm using the same session object to track these but you can certain use a separate one if you prefer. In the SessionManager.cfc add this method:

```
<cfscript>
public void function logPageAction( required string pageAction, numeric maxPages=40 ) {
if ( arrayLen(session.pageTracker) IS 0 OR session.pageTracker[1] IS NOT arguments.pageAction ) {
lock scope="session" type="exclusive" timeout="10" {
     if ( arrayLen(session.pageTracker) LT arguments.maxPages ) {
        arrayPrepend(session.pageTracker,arguments.pageAction);
     } else {
        arrayDeleteAt(session.pageTracker,arrayLen(session.pageTracker));
        arrayPrepend(session.pageTracker,arguments.pageAction);
        }
     }
     }
}
</cfscript>
```

In the remote service layer, add this method to send data to the logger:

```
<cfscript>
remote function updatePageTracker( string pageAction = '') {
if (NOT len(arguments.pageAction) ) {
    arguments.pageAction = cgi.QUERY_STRING;
}
sessionManager.logPageAction(arguments.pageAction);
return true;
}
</cfscript>
```

And finally here's our JS code to send a logging entry to the session manager. Notice we use a promise so that we can make sure the logging call completes before moving to the next Ajax call or page load.

```
<InvalidTag>
function updatePageTracker(pageAction){
    pagetrackerJQXHR = $.ajax({
        url: '/remote/remoteSessionManager.cfc',
        data: {
            method: 'updatePageTracker',
            pageAction: pageAction,
            returnFormat: 'json'
        }
    }).done( function(response) {
        // success
        return response;
    });
}
</script>
```

So just include the .JS file with that code into your header and you can send any JS action to the tracker:

```
updatePageTracker('Add Item to Shopping Cart');
```

So now, when an error occurs in the site, or we purposely throw an error to debug a particular issue in the code that is occurring, we now will have included in the session data a nice list of the last 20 page requests and actions that the user made prior to the error occurring. You can tweak the number of actions to log, fewer or more as works best for your site, but 40 seems to be a number that works well for us to get a sufficient amount of information on the user's recent activity. We'll add in the Ajax tracking in the next blog article, which gives us even more detailed information on the user interaction on the site.

Most of the rest of the updates I've made to the error handler relate to logging of errors so let's move on to the first option for this. At the time I wrote my original post, we were using the excellent bug tracking system for ColdFusion, BugLogHQ. This is an open source project that is easy

to set up and is very feature-rich. If you are on a budget (or even if you aren't!) I highly recommend it. We now use a different tracker which we'll look at next time, but for this article, we'll dive into BugLogHQ and how you can set up and use that with your error handler as it's a great option for CF developers that don't want to use a paid service.

Once you have installed and set up the BugLogHQ application (download from www.bugloghq.com), and started up the REST service, you can configure it for your application. I recommend initializing it in your ApplicationStart() method and loading it into application scope. The init() method for Buglog requires the URL to the REST service, the API key, and optionally a list of "secure" variables you want to strip from your error messages. So you may want to include things like user passwords, encryption keys, etc. BugLog can also take an argument for "sanitizeExpressions" which is an array of RegEx expressions to use for sanitizing. According to the BugLog docs, "Out of the box the BugLog client excludes any Javascript code, credit card numbers and S3 passwords".

```
<cfscript>
variables.securevars="CFID,CFTOKEN,JSESSIONID,password,,authkey,SessionID,URLToken,encryptkey ";
application.bugLogService = createObject("component",
"buglog.client.bugLogService").init(bugLogListener='http://buglog.servername.com/listeners/bugLogListenerREST.cfm',sensitiveFieldNames=variables.securevars,
apikey="XXXXX");
</cfscript>
```

You may want to wrap that code block in a try/catch in case there is some problem with starting up the service.

To send your errors to BugLog, you simply call the notify method with the error name and error object:

```
<cfscript>

catch (any e) {
    Application.BugLogService.notifyService(e.message, e);
}

</cfscript>
```

However, what if you want to include all that diagnostic scope information that we built into the error handler, and not just error information and basic CGI data? It would be nice for instance to have that page tracker in the session we just added. Instead of calling BugLog in catch blocks, we want to include it into the error handler and send the sanitized scope data. Additionally, BugLog lets us send an error type that we can use to filter our errors on -- a perfect use for our new error type setting that we added earlier. In addition, the error handler has been designed to reduce the chance of an error throwing out hundreds or thousands of similar errors that can overwhelm an email server... or the in case of BugLog, the database handling the error logging (see the previous blog article for more information on doing this). So instead of calling BugLog in the catch block, we'll include our error handler in the catch and move the BugLog notify into the section where we check for previous instances of an error, and send email notifications for errors. We'll want to make sure that the service has been loaded into application scope to use it.

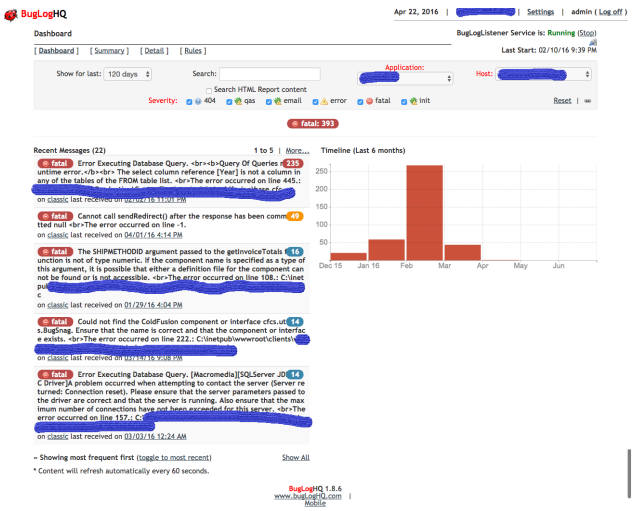
```
<cfif structKeyExists(application,"bugLogService")>
<cfset application.bugLogService.notifyService(localVars.errormess, localVars.errorData, localVars.strScopes, request.errorType)>
</cfif>
```

For errors that in my handler that I exclude from email due to them occurring regularly (database timeout errors for instance are one thing that we get regularly and are beyond our control so aren't useful to get notifications of) I can use BugLog to still log these errors to have an idea of how often they are occurring and in case I do need to review them.

In addition to logging your CF errors, you can use BugLog to log other kinds of errors. Here's a call I added to the top of our custom 404 page to log these errors to BugLog so we can easily view and fix missing file and page errors, without having to use something else like Google Analytics to find them. We include the CGI scope to include additional information such as the referring page.

```
<cfif structKeyExists( application, "bugLogService" ) >
<cfset application.bugLogService.notifyService("Page Not Found - " & cgi.script_name & "?" & cgi.query_string, CGI, URL, "404")>
</cfif>
</cfset>
```

So here's what our BugLogHQ Dashboard looks like (I apologize for the redactions but error information is a treasure trove for hackers). BugLog gives us filters for the different types of errors we log as well as for host and application names, and a text search which can drill into the full text of the errors as well. Each error has a count for number of times we logged it, and you can click on the error to drill down into the details. BugLog itself has very comprehensive email notification settings so is great for team situations where you may want to add and remove people from notifications of errors on specific projects versus manually configuring emailing addresses in your error handler. You can also subscribe to a daily summary of errors.



And here's a sample detailed view of an error. As you can see, BugLog gives us a really nicely formatted view of this nested ColdFusion structure of all our data scopes.

**Bug #167132 : Cannot call sendRedirect() after the response has been committed null
The error occurred on line -1.**

BugLogListener Service is: Running (Stop)
Last Start: 02/10/16 9:39 PM

[Dashboard] [Summary] [Detail] [Rules]

Send to email | Delete | Create rule | fatal | Application: [redacted] | Host: [redacted]

Date/Time:	04/01/16 4:14 PM
User Agent:	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_5) AppleWebKit/601.5.17 (KHTML, like Gecko) Version/9.1 Safari/601.5.17
Template Path:	C:\inetpub\[redacted]
Exception Message:	Cannot call sendRedirect() after the response has been committed
Exception Detail:	

Stats
- This is the **first time** this message has occurred in the last 24 hours.
- This bug has occurred 127 times since 11/28/15 10:07 AM
- The previous time this bug was reported was on 03/09/16 7:34 PM

Host Distribution:

CWHServer1555	68 (54%)
CWHServer1554	59 (46%)

HTML Report

Exception Summary

Application: [redacted]
Host: [redacted]
URL: [redacted]
Server Date/Time: [redacted]
Message: Cannot call sendRedirect() after the response has been committed
Type: coldfusion.runtime.CfErrorWrapper
Detail: [redacted]
Script Name (CGI): /[redacted]
User Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_5) AppleWebKit/601.5.17 (KHTML, like Gecko) Version/9.1 Safari/601.5.17
Remote Address: 172.242.29.183
Referrer: [redacted]
Query String:
Request Method: POST
ColdFusion ID: [SESSION] CFID = - ; CFTOKEN = — ; SESSIONID = C0E826518F5260D2558C1132B853E2DB.Production
[CLIENT] CFID = — ; CFTOKEN = —
[COOKIES] CFID = — ; CFTOKEN = — ; JSessionID = C0E826518F5260D2558C1132B853E2DB.Production

Exception Info

Browser: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_5) AppleWebKit/601.5.17 (KHTML, like Gecko) Version/9.1 Safari/601.5.17
Mailto:
Suppressed: *Empty array*
StackTrace:

```
java.lang.IllegalStateException: Cannot call sendRedirect() after the response has been committed
	at org.apache.catalina.connector.ResponseFacade.sendRedirect(ResponseFacade.java:482)
	at javax.servlet.http.HttpServletResponseWrapper.sendRedirect(HttpServletResponseWrapper.java:137)
	at javax.servlet.http.HttpServletResponseWrapper.sendRedirect(HttpServletResponseWrapper.java:137)
	at javax.servlet.http.HttpServletResponseWrapper.sendRedirect(HttpServletResponseWrapper.java:137)
	at coldfusion.jsp.HttpServletResponseWrapper.sendRedirect(HttpServletResponseWrapper.java:65)
	at coldfusion.filter.ComponentFilter.invoke(ComponentFilter.java:161)
	at coldfusion.filter.ApplicationFilter.invoke(ApplicationFilter.java:480)
	at coldfusion.filter.RequestMonitorFilter.invoke(RequestMonitorFilter.java:42)
	at coldfusion.filter.MonitoringFilter.invoke(MonitoringFilter.java:40)
	at coldfusion.filter.PathFilter.invoke(PathFilter.java:142)
	at coldfusion.filter.ExceptionFilter.invoke(ExceptionFilter.java:94)
	at coldfusion.filter.ClientScopePersistenceFilter.invoke(ClientScopePersistenceFilter.java:28)
	at coldfusion.filter.BrowserFilter.invoke(BrowserFilter.java:38)
	at coldfusion.filter.NoCacheFilter.invoke(NoCacheFilter.java:58)
	at coldfusion.filter.GlobalsFilter.invoke(GlobalsFilter.java:38)
	at coldfusion.filter.DatasourceFilter.invoke(DatasourceFilter.java:22)
	at coldfusion.xml.rpc.CFCServlet.invoke(CFCServlet.java:156)
	at coldfusion.xml.rpc.CFCServlet.doPost(CFCServlet.java:348)
	at javax.servlet.http.HttpServlet.service(HttpServlet.java:650)
	at javax.servlet.http.HttpServlet.service(HttpServlet.java:731)
	at coldfusion.bootstrap.BootstrapServlet.service(BootstrapServlet.java:89)
	at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:303)
	at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:208)
	at coldfusion.monitor.event.MonitoringServletFilter.doFilter(MonitoringServletFilter.java:42)
	at coldfusion.bootstrap.BootstrapFilter.doFilter(BootstrapFilter.java:46)
	at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:241)
	at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:208)
	at sun.reflect.GeneratedMethodAccessor77.invoke(Unknown Source)
	at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
	at java.lang.reflect.Method.invoke(Method.java:497)
	at com.intergral.fusionreactor.j2ee.filterchain.WrappedFilterChain.doFilter(WrappedFilterChain.java:97)
	at com.intergral.fusionreactor.j2ee.filter.FusionReactorRequestHandler.doNext(FusionReactorRequestHandler.java:
472)
	at com.intergral.fusionreactor.j2ee.filter.FusionReactorRequestHandler.doHttpServletRequest(FusionReactorReques
tHandler.java:312)
	at com.intergral.fusionreactor.j2ee.filter.FusionReactorRequestHandler.doFusionRequest(FusionReactorRequestHand
ler.java:192)
	at com.intergral.fusionreactor.j2ee.filter.FusionReactorRequestHandler.handle(FusionReactorRequestHandler.java:
507)
	at com.intergral.fusionreactor.j2ee.filter.FusionReactorCoreFilter.doFilter(FusionReactorCoreFilter.java:36)
	at sun.reflect.GeneratedMethodAccessor76.invoke(Unknown Source)
	at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
	at java.lang.reflect.Method.invoke(Method.java:497)
	at com.intergral.fusionreactor.j2ee.filterchain.WrappedFilterChain.doFilter(WrappedFilterChain.java:79)
	at sun.reflect.GeneratedMethodAccessor75.invoke(Unknown Source)
	at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
	at java.lang.reflect.Method.invoke(Method.java:497)
	at com.intergral.fusionreactor.agent.filter.FusionReactorStaticFilter.doFilter(FusionReactorStaticFilter.java:5
3)
	at com.intergral.fusionreactor.agent.pointcuts.NewFilterChainPointCut$1.invoke(NewFilterChainPointCut.java:41)
	at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java)
	at org.apache.catalina.core.StandardWrapperValve.invoke(StandardWrapperValve.java:220)
	at org.apache.catalina.core.StandardContextValve.invoke(StandardContextValve.java:122)
	at org.apache.catalina.authenticator.AuthenticatorBase.invoke(AuthenticatorBase.java:505)
	at org.apache.catalina.core.StandardHostValve.invoke(StandardHostValve.java:170)
	at org.apache.catalina.valves.ErrorReportValve.invoke(ErrorReportValve.java:103)
	at org.apache.catalina.core.StandardEngineValve.invoke(StandardEngineValve.java:116)
	at org.apache.catalina.connector.CoyoteAdapter.service(CoyoteAdapter.java:437)
	at org.apache.coyote.ajp.AjpProcessor.process(AjpProcessor.java:197)
	at org.apache.coyote.AbstractProtocol$AbstractConnectionHandler.process(AbstractProtocol.java:625)
	at org.apache.tomcat.util.net.JIoEndpoint$SocketProcessor.run(JIoEndpoint.java:316)
	at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1142)
	at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:617)
	at org.apache.tomcat.util.threads.TaskThread$WrappingRunnable.run(TaskThread.java:61)
	at java.lang.Thread.run(Thread.java:745)
```

HTTPReferer: [redacted]
DateTime: [redacted]
RemoteAddress: 172.242.29.183
Template: [redacted]
RootCause: [redacted]

struct

Message	Cannot call sendRedirect() after the response has been committed
StackTrace	java.lang.IllegalStateException: Cannot call sendRedirect() after the response has been committed at org.apache.catalina.connector.ResponseFacade.sendRedirect(ResponseFacade.java:482) at javax.servlet.http.HttpServletResponseWrapper.sendRedirect(HttpServletResponseWrapper.java:137) at javax.servlet.http.HttpServletResponseWrapper.sendRedirect(HttpServletResponseWrapper.java:137) at javax.servlet.http.HttpServletResponseWrapper.sendRedirect(HttpServletResponseWrapper.java:137) at coldfusion.jsp.HttpServletResponseWrapper.sendRedirect(HttpServletResponseWrapper.java:65) at coldfusion.filter.ComponentFilter.invoke(ComponentFilter.java:161) at coldfusion.filter.ApplicationFilter.invoke(ApplicationFilter.java:480) at coldfusion.filter.RequestMonitorFilter.invoke(RequestMonitorFilter.java:42) at coldfusion.filter.MonitoringFilter.invoke(MonitoringFilter.java:40) at coldfusion.filter.PathFilter.invoke(PathFilter.java:142) at coldfusion.filter.ExceptionFilter.invoke(ExceptionFilter.java:94) at coldfusion.filter.ClientScopePersistenceFilter.invoke(ClientScopePersistenceFilter.java:28) at coldfusion.filter.BrowserFilter.invoke(BrowserFilter.java:38) at coldfusion.filter.NoCacheFilter.invoke(NoCacheFilter.java:58) at coldfusion.filter.GlobalsFilter.invoke(GlobalsFilter.java:38) at coldfusion.filter.DatasourceFilter.invoke(DatasourceFilter.java:22) at coldfusion.xml.rpc.CFCServlet.invoke(CFCServlet.java:156) at coldfusion.xml.rpc.CFCServlet.doPost(CFCServlet.java:348) at javax.servlet.http.HttpServlet.service(HttpServlet.java:650) at javax.servlet.http.HttpServlet.service(HttpServlet.java:731) at coldfusion.bootstrap.BootstrapServlet.service(BootstrapServlet.java:89) at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:303) at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:208) at coldfusion.monitor.event.MonitoringServletFilter.doFilter(MonitoringServletFilter.java:42) at coldfusion.bootstrap.BootstrapFilter.doFilter(BootstrapFilter.java:46) at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:241) at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:208) at sun.reflect.GeneratedMethodAccessor77.invoke(Unknown Source) at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43) at java.lang.reflect.Method.invoke(Method.java:497) at com.intergral.fusionreactor.j2ee.filterchain.WrappedFilterChain.doFilter(WrappedFilterChain.java:97) at com.intergral.fusionreactor.j2ee.filter.FusionReactorRequestHandler.doNext(FusionReactorRequestHandler.java:472) at com.intergral.fusionreactor.j2ee.filter.FusionReactorRequestHandler.doHttpServletRequest(FusionReactorRequestHandler.java:312) at com.intergral.fusionreactor.j2ee.filter.FusionReactorRequestHandler.doFusionRequest(FusionReactorRequestHandler.java:192) at com.intergral.fusionreactor.j2ee.filter.FusionReactorRequestHandler.handle(FusionReactorRequestHandler.java:507) at com.intergral.fusionreactor.j2ee.filter.FusionReactorCoreFilter.doFilter(FusionReactorCoreFilter.java:36) at sun.reflect.GeneratedMethodAccessor76.invoke(Unknown Source) at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43) at java.lang.reflect.Method.invoke(Method.java:497) at com.intergral.fusionreactor.j2ee.filterchain.WrappedFilterChain.doFilter(WrappedFilterChain.java:79) at sun.reflect.GeneratedMethodAccessor75.invoke(Unknown Source) at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43) at java.lang.reflect.Method.invoke(Method.java:497) at com.intergral.fusionreactor.agent.filter.FusionReactorStaticFilter.doFilter(FusionReactorStaticFilter.java:53) at com.intergral.fusionreactor.agent.pointcuts.NewFilterChainPointCut$1.invoke(NewFilterChainPointCut.java:41) at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java) at org.apache.catalina.core.StandardWrapperValve.invoke(StandardWrapperValve.java:220) at org.apache.catalina.core.StandardContextValve.invoke(StandardContextValve.java:122) at org.apache.catalina.authenticator.AuthenticatorBase.invoke(AuthenticatorBase.java:505) at org.apache.catalina.core.StandardHostValve.invoke(StandardHostValve.java:170) at org.apache.catalina.valves.ErrorReportValve.invoke(ErrorReportValve.java:103) at org.apache.catalina.core.StandardEngineValve.invoke(StandardEngineValve.java:116) at org.apache.catalina.connector.CoyoteAdapter.service(CoyoteAdapter.java:437) at org.apache.coyote.ajp.AjpProcessor.process(AjpProcessor.java:197) at org.apache.coyote.AbstractProtocol$AbstractConnectionHandler.process(AbstractProtocol.java:625) at org.apache.tomcat.util.net.JIoEndpoint$SocketProcessor.run(JIoEndpoint.java:316) at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1142) at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:617) at org.apache.tomcat.util.threads.TaskThread$WrappingRunnable.run(TaskThread.java:61) at java.lang.Thread.run(Thread.java:745)
Suppressed	array [empty]
TagContext	array [empty]
Type	java.lang.IllegalStateException

QueryString:
Diagnostics: Cannot call sendRedirect() after the response has been committed null
The error occurred on line -1.

Additional Info

JavaScript code removed for security

struct

CGI

struct

AUTH_PASSWORD	*** SENSITIVE DATA REMOVED ***
AUTH_TYPE	[empty string]
AUTH_USER	*** SENSITIVE DATA REMOVED ***
CERT_COOKIE	[empty string]
CERT_FLAGS	[empty string]
CERT_ISSUER	[empty string]
CERT_KEYSIZE	[empty string]
CERT_SECRETKEYSIZE	[empty string]
CERT_SERIALNUMBER	[empty string]
CERT_SERVER_ISSUER	[empty string]
CERT_SERVER_SUBJECT	[empty string]
CERT_SUBJECT	[empty string]
CF_TEMPLATE_PATH	[redacted]
CONTENT_LENGTH	52
CONTENT_TYPE	application/x-www-form-urlencoded; charset=UTF-8
CONTEXT_PATH	[empty string]
GATEWAY_INTERFACE	CGI/1.1
HTTPS	off
HTTPS_KEYSIZE	[empty string]
HTTPS_SECRETKEYSIZE	[empty string]
HTTPS_SERVER_ISSUER	[empty string]
HTTPS_SERVER_SUBJECT	[empty string]
HTTP_ACCEPT	application/json, text/javascript, */*; q=0.01
HTTP_ACCEPT_ENCODING	gzip


HTTP_ACCEPT_LANGUAGE en-us
HTTP_CONNECTION keep-alive
HTTP_COOKIE *** SENSITIVE DATA REMOVED ***
HTTP_HOST [redacted]
HTTP_REFERER [redacted]
HTTP_URL [redacted]
HTTP_USER_AGENT Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_5) AppleWebKit/601.5.17 (KHTML, like Gecko) Version/9.1 Safari/601.5.17
LOCAL_ADDR www.classicindustries.com
PATH_INFO [empty string]
PATH_TRANSLATED [redacted]
QUERY_STRING [empty string]
REMOTE_ADDR 172.242.29.183
REMOTE_HOST 172.242.29.183
REMOTE_USER [empty string]
REQUEST_METHOD POST
SCRIPT_NAME [redacted]
SERVER_NAME [redacted]
SERVER_PORT 80
SERVER_PORT_SECURE 0
SERVER_PROTOCOL HTTP/1.1
SERVER_SOFTWARE Microsoft-IIS/7.5
WEB_SERVER_API [empty string]
CLIENT struct [empty]
COOKIE struct
CATALOGID 3
CUSTOMERID 0
JSESSIONID *** SENSITIVE DATA REMOVED ***
LB-Persist 1347947018.20480.0000
LISTAS 1
RESULTSPERPAGE 20
SEARCHPARAMETERS [redacted]
SEARCHYEAR 56
SORTBY 0
VEHICLEMODELID 0
__atuvc 1|13
__atuvs 56fed605ec68c75e000
__hssc 78576587.2.1459541470807
__hssrc 1
__hstc 78576587.6dc5b177e562a97f0ddd11459e5d11db.1459541470807.1459541470807.1459541470807.1
_ga GA1.2.2036258767.1459541467
_gat 1
bhtglocalcookperstest 1
hsfirstvisit [redacted]
hubspotutk 6dc5b177e562a97f0ddd11459e5d11db
s_cc true
s_eVar26 New
s_fid 649C9FB91A477100-032A10F015D1B85A
s_invisit true
s_lv 1459541620581
s_lv_s First Visit
s_nr 1459541620577-New
s_sq [[B]]
s_vnum 1462075200359&vn=1
unencrypted_CUSTOMERID 0
FILE struct [empty]
FORM struct [empty]
REQUEST struct
__FusionReactor YES
cfdumpinited false
errortime 1459541617555
errortype [redacted]
template [redacted]
webpageads struct
checkoutShippingFlg query
checkoutSummaryFlg query
homepageFeatureLeftFlg query
homepageFeatureRightFlg query
homepageHeaderFlg query
pDPPageFlg query
searchPageFlg query
siteFooterFlg query
siteHeaderFlg query
ALTTEXT CHECKOUTSHIPPINGFLG CHECKOUTSUMMARYFLG ENDDATETIME HOMEPAGEFEATURELEFTFLG
1 Up To 25% OFF!* Online Orders Over $299 0 0 2016-04-03 23:59:00.0 0 0
SESSION struct
ajaxtracker array - Top 2 of 2 rows
1 struct
AJAXREQUEST [redacted]
METHODARGS struct
CATALOGID 3
MODELID 0
SEARCHTERM [empty string]
YEARID 56
METHODNAME [redacted]
REQUESTTIME 01-Apr-16-01:11 PM
2 struct
AJAXREQUEST [redacted]
METHODARGS struct
CUSTPRICETYPEID 1
LSTPRODUCTIDS [redacted]
METHODNAME [redacted]
REQUESTTIME 01-Apr-16-01:11 PM
carttotals struct
CARTTOTALPRODUCTS 0
TOTALDOLLARAMOUNT 0
catalog struct
ID 3
NAME firebird
customerid 0
custpricetypeid 1
email [empty string]
emailpath [redacted]
imagepath [redacted]
lastsearch struct
NUMTIMES 1
SEARCHURL [redacted]
TIMERAN {ts '2016-04-01 13:10:51'}
model struct
FULLNAME [empty string]
ID 0
NAME [empty string]
pagetracker array - Top 3 of 3 rows
1 [redacted]
2 [redacted]
3 [redacted]
pricetypeid 1
searchterm [empty string]
secureurl [redacted]
sessionid *** SENSITIVE DATA REMOVED ***
sessionversion 26
shoppingcartid 0
sitedomain [redacted]
sitename [redacted]
siteversion 4.4
template [redacted]
url [redacted]
urltoken *** SENSITIVE DATA REMOVED ***
user_ip 172.242.29.183
year struct
ID 56
YEARSTRING 1967
URL struct [empty]
VARIABLES struct
ISCFC function IsCFC
Arguments:
Name Required Type Default
objectToCheck Optional Any
ReturnType: Any
Roles:
Access: public
Output:
DisplayName:
Hint: Returns a boolean for whether a CF variable is a CFC instance.
Description:
Return To Log

So that's BugLogHQ. Next time we'll look at doing comprehensive logging for Ajax requests and look at more advanced tool for logging both Javascript and ColdFusion errors.