Better Exception Handling in ColdFusion

Posted At : October 26, 2015 10:00 AM | Posted By : Steve
Related Categories: ColdFusion

We have been using basically the same exception handling strategy for several years. It has worked pretty well in all of that time, but we recently decided to switch it up. Our previous exception handling system (the one that we have been using for years) made sure not to display valuable information to users, but did send us an email with the pertinent information.

This made for a system that was reasonably secure (in that it didn't share any sensitive data on screen), but still allowed for us to quickly find out about any problems so that we could take care of them.

The system, however, has had two pretty big drawbacks. First, if ColdFusion is unable to send email then we don't get any messages. Second, if an exception ever happens in a high-traffic system then we get positively deluged with messages. This doesn't happen very often, but I have had server changes mess things up and wake up with several hundred email messages before.

You might be thinking that it is a good thing that we get a lot of messages for a major problem, but really it isn't. It ties up the mail server and can tie up a mail client at a time when you really want to read the message.

So, our new system has a few changes over the old one. For one thing, it sends notifications both to Slack and via email (with more details in the email). For another, it rapidly scales down the notifications.

I'll cover our Slack integration later. Just know that the Slack API is crazy-easy to implement (kudos to them on that).

For now, however, I want to cover how we scale down the message volume without losing critical data.

The first thing that we need to do is figure out the message for the error. This seems straightforward, but sometimes ColdFusion doesn't put the most relevant message at the top of the structure.

As you can see, this is in a custom tag to which we pass the full exception (using either Error="#Error#" or Error="#CFCATCH#" depending on the usage).

```
<cfif StructKeyExists(Attributes.Error,"Cause")>
   <cfset message = Attributes.Error.Cause.Message>
<cfelseif StructKeyExists(Attributes.Error,"Message")>
   <cfset message = Attributes.Error.Message>
<cfelse>
   <cfset message = "Unknown">
</cfif>
```

Basically, we are just getting the error message from the "Cause" key if it exists. Otherwise we are just getting the message for the exception.

Once we get the message, then we need to decide a unique key for the exception. I use a Hash() of the error message (or a CreateUUID() if there is none to make sure I hear about every one of those.

```
<cfif StructKeyExists(Attributes.Error,"Cause")>
   <cfset key = Hash(Attributes.Error.Cause.Message)>
   <cfset message = Attributes.Error.Cause.Message>
<cfelseif StructKeyExists(Attributes.Error,"Message")>
   <cfset key = Hash(Attributes.Error.Message)>
   <cfset message = Attributes.Error.Message>
<cfelse>
   <cfset key = CreateUUID()>
   <cfset message = "Unknown">
```

```
</cfif>
```

Next, I use an Application scoped variable to keep track of exceptions.

I use a structure and keep track of a few valuable data points about the exception. Here it is in an "sAlerts" structure (for reasons that will be clear in a later entry), but you could name it anything.

```
<!--- Get the structure key started for this alert message. --->
<cfif StructKeyExists(Application.sAlerts,Attributes.key)>
   <!--- Increase the number of times the alert has occurred. --->
   <cfset Application.sAlerts[Attributes.key]["TimeLast"] = now()>
   <cfset Application.sAlerts[Attributes.key]["Times"] = Application.sAlerts[Attributes.key]["Times"] + 1>
<cfelse>
   <cfset Application.sAlerts[Attributes.key] = {}>
   <cfset Application.sAlerts[Attributes.key]["Message"] = Attributes["message"]>
   <cfset Application.sAlerts[Attributes.key]["TimeFirst"] = now()>
   <cfset Application.sAlerts[Attributes.key]["TimeLast"] = now()>
   <cfset Application.sAlerts[Attributes.key]["Times"] = 1>
</cfif>
<cfset NumMessages = Application.sAlerts[Attributes.key]["Times"]>
```

The key here is that we know a bit about exceptions for this key. Now we can update the key with this exception. Above this block of code, we have another to clear the data out every two hours.

```
<cfparam name="Application.sAlerts" default="#StructNew()#">
<cfparam name="Application.AlertsLastCleaned" default="#now()#" type="date">

<!--- Clean out old alerts if it hasn't been done in the last hour --->
<cfif DateDiff("h",Application.AlertsLastCleaned,now()) GTE 1>
   <cfloop collection="#Application.sAlerts#" item="currkey">
      <!--- Remove any error that hasn't happened for more than two hours. We'll reset the count at that point --->
      <cfif DateDiff("h",Application.sAlerts[currkey]["TimeLast"],now()) GTE 2>
         <cfset StructDelete(Application.sAlerts,currkey)>
      </cfif>
   </cfloop>
   <cfset Application.AlertsLastCleaned = now()>
</cfif>
```

If it has been more than two hours since the exception has occurred then we start the count over.

The upshot here is that NumMessages holds the number of times the exception has occurred in the last two hours.

```
<cfif Log10(NumMessages) EQ Round(Log10(NumMessages))>
```

What exactly you do in this conditional is up to you, but this is where you send notification (in our case via both email and Slack). This means that you only send notifications about an exception each time it has happened ten times more than for the last notification (so, the first, 10th, 100th, 1,000th, 10,000th etcetera). Make sure to include #NumMessages# in the notifications.

This strategy ensures that we find out about all errors right away (unless both email and Slack are down – maybe we should add SMS as well?), but we don't get deluged with repeat information about any errors.