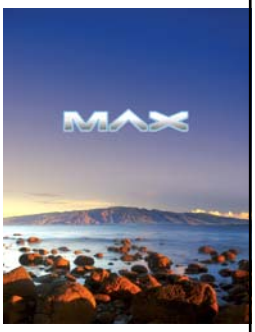
# MAX 2006
# Beyond Boundaries




Jason Delmore

Developing ColdFusion-Java Hybrid Applications

October 24th 2006




## Biography

Jason J Delmore
Senior Product Manager, ColdFusion
Adobe Systems Inc.

- Began developing complex applications with ColdFusion in 1999.
- Architected and led the development of several enterprise level applications primarily focused on
  - Data collection and management
  - Self-service Reporting
  - E-commerce
  - Integration
- Very experienced in leveraging Java from within ColdFusion.




## Agenda

- Overview
- Why Use ColdFusion and Java Together?
- Enhancing ColdFusion with Java
  - Using Java CFX Tags
  - Calling Java Objects (APIs, Classes, EJBs, etc.)
  - Integrating JSPs & Servlets
  - Using Custom JSP Tags
  - Calling ColdFusion from Java classes?
- Limitations of Java in ColdFusion
- Pulling It All Together
- Some Suggestions
- Questions and Answers
- Where to Find Out More




## Overview

- ColdFusion is a productivity layer **built on the strong foundation of J2EE**.
- ColdFusion is a **Rapid Application Development** (RAD) environment with a rich feature set already built-in (Flex, Reporting, Document creation, etc).
- Java has advantages in **scalability** and has become the language of choice for **open source projects**.
- ColdFusion applications can **deliver the best of both worlds**: the strength of Java and the productivity of ColdFusion.
- This session will cover the interaction between ColdFusion and Java, and strategies for building powerful hybrid applications.




## Why use ColdFusion and Java Together?

- **The problem**
  - Although highly productive and easy to develop with, ColdFusion alone may not provide all of the functionality required to develop an enterprise-scale application
  - Java applications typically take longer to develop and many tasks that are trivial to do in ColdFusion are far more complex in a standard Java development environment
- **The solution**
  - ColdFusion provides quick and easy application development
  - Java provides a robust solution set for developing enterprise-scale applications
  - Hybrid applications leverage the ease of ColdFusion and the power of Java




## Enhancing ColdFusion with Java

- Leveraging the Java underpinnings of ColdFusion is easier than you think
- ColdFusion has been able to leverage Java since ColdFusion 4.5
- It is far easier to leverage Java today, with many more capabilities and methods to do so!

## Using Java CFX Tags

- ColdFusion Extension (CFX) tags were the first tool for integrating ColdFusion with Java
  - Java CFX API (cfx.jar) available for development
  - Allowed developers to extend the functionality of ColdFusion pages beyond CFML




## Using Java CFX Tags

- Example - HelloColdFusion
  - Let's write some simple code that uses the CFX API.

```
// Hello ColdFusion

import com.allaire.cfx.* ;

public class MyHelloColdFusion implements CustomTag {
   public void processRequest( Request request, Response response )
      throws Exception {
      String strName = request.getAttribute( "NAME" ) ;
      response.write( "Hello, " + strName ) ;
   }
}
```




## Using Java CFX Tags

- Example - HelloColdFusion (Continued)
  - Compile the code
    `javac -classpath C:\CFusionMX7\wwwroot\WEB-INF\lib\cfx.jar MyHelloColdFusion.java`
  - Register the CFX in the CF Administrator





## Using Java CFX Tags

- Example - HelloColdFusion (Continued)
  - Create a CFM that uses the tag

```
<cfx_HelloColdFusion name="Inigo Montoya"/>
```

  - Output is

    "Hello, Inigo Montoya"




## Using Java CFX Tags

- Limitations of Java CFXs (there are many!)
  - They are **not standalone applications**, they are created and invoked by the ColdFusion server
    - This makes **debugging more difficult**
    - This also makes the **code less re-useable**
  - They **must be registered within the ColdFusion Administrator**
  - They have **poor datatype support** (only strings and queries)
  - They have **no protection**… a poorly written CFX can crash the server




## Calling Java Objects

- J2SE & J2EE APIs are available in ColdFusion by default
  - Accessed using <CFOBJECT>, <CFINVOKE> or CreateObject()
  - Static members available by default
  - Instance members available after calling reserved init() method, or
  - Instance members available after calling non-static method without calling init() (implicitly created with default constructor)

## Calling Java Objects

- **<CFOBJECT>**
  - Tag to load a Java classes
- **<CFINVOKE>**
  - Tag to invoke methods
  - If you do not create an object before calling a function with CFINVOKE, the tag will create a class object for just that method call and then throw the object away.
- **CreateObject()**
  - Same as CFOBJECT but can be used in CFSCRIPT or CFSET
  - ***Just My Opinion:*** *Always use CreateObject()!*




## Calling Java Objects

- Example - StringBuffer
  - Creating a StringBuffer object

```
<!--- instantiate a StringBuffer class --->
<cfset StringBuffer =
  createObject("java","java.lang.StringBuffer")/>

<!--- create a StringBuffer instance, analog to java new() --->
<cfset StringBuffer.init("Hello ")/>

<!--- append to the buffer --->
<cfset StringBuffer.append("World!")/>
<cfoutput>#StringBuffer.toString()#</cfoutput>

<!--- reverse the buffer --->
<cfset StringBuffer.reverse()/>
<cfoutput>#StringBuffer.toString()#</cfoutput>
```




## Calling Java Objects

- Example - StringBuffer
  - Output from example

**Hello World!**
**!dlroW olleH**

- **Hidden feature:** A ColdFusion String extends java.lang.String… and so it responds to all of the functions for java.lang.String!




## Calling Java Objects

- Demonstration – Read file with an input stream - FileReader.cfm
  - Reading from a large file one line at a time

```
<cfset BufferSize = "8192"/>
<cfset FileInputStream = CreateObject("java","java.io.FileInputStream")/>
<cfset FileInputStream.init("C:\CFusionMX7\wwwroot\examples\fileToRead.txt")/>
<cfset InputStreamReader = CreateObject("java","java.io.InputStreamReader")/>
<cfset InputStreamReader.init(FileInputStream,  "UTF-8")/>
<cfset BufferedReader = CreateObject("java","java.io.BufferedReader")/>
<cfset BufferedReader.init(InputStreamReader, BufferSize)/>
<cfset BufferedReader.mark(BufferSize)/>
<cfloop condition="BufferedReader.read() neq -1">
  <cfset BufferedReader.reset()/>
  <cfoutput>#BufferedReader.readLine()#</cfoutput><br/>
  <cfset BufferedReader.mark(BufferSize)/>
</cfloop>
<cfset BufferedReader.close()/>
<cfset InputStreamReader.close()/>
<cfset FileInputStream.close()/>
```




## Calling Java Objects

- Demonstration – Serialize a ColdFusion Object - SerializeDeserialize.cfm
  - Serialize and Deserialize ColdFusion objects (Not CFCs)
    - Struct
    - Array
    - Query
    - … Any object that implements the java.io.serializable interface




## Calling Java Objects

- An Enterprise Java Bean (EJB) is a java object that is managed by the container in which it is hosted, and provides
  - Scalability features (clustering, pooling, caching, etc)
  - Transaction management
  - Fine-grained security
  - And more…
- EJBs in ColdFusion
  - ColdFusion cannot host EJBs; however, it can use EJBs hosted by the J2EE server it is deployed on

## Calling Java Objects

- Example
  - Calling an EJB from a ColdFusion page

```
<!--- Create the InitialContext --->
<cfset context = CreateObject("java",
  "javax.naming.InitialContext")/>

<!--- Locate the EJB and obtain a reference --->
<cfset home = context.lookup("Employee")/>

<!--- Create an instance of the EJB --->
<cfset employee = home.create()/>

<!--- Get the name for a given employee --->
<cfset employeeName= employee.getName("12345")/>
```




## Calling Java Objects

- Any Java API or Java class can also be used in ColdFusion
  - Classes must be available to either the JVM classpath or the ColdFusion classpath
  - By default CF ClassLoader will load
    - *cfroot*/lib
    - *cfroot*/gateway/lib
  - The J2EE container will load
    - *cfroot*/WEB-INF/lib
    - *cfroot*/WEB-INF/classes
  - ***My Suggestion:*** *Add you own directory to the CF Classpath*




## Calling Java Objects

- Adding your own classpath to ColdFusion
  - For ColdFusion Standalone:
    - In the ColdFusion administrator, click on "Java and JVM", then add the absolute paths to your jar or class files in the "Class Path" field.
  - For JRun Deployment:
    - Open up the JRun Management Console and under the default server, click on settings, then add your classes to the class path list.
  - For other app servers or when in doubt, you can edit *cfroot*/runtime/bin/jvm.config.
    `# Arguments to VM`
    `java.args= …` -Dcoldfusion.classPath={application.home}/../lib/updates
  - Add your path to the coldfusion.classPath not the java.class.path.




## Calling Java Objects

- Create a CFC that makes your calls to the Java API
- Create a custom tag for accessing that CFC – its much easier than you think
  - Just drop a CFM defining the tags actions in the *cfroot*/CustomTags directory
- Now you can call it via tag syntax, or call the CFC directly! (CFC syntax can be used in CFSCRIPT)




## Calling Java Objects

- Demonstration
  - Leveraging JExcel API to build Excel Documents in ColdFusion
  - Files Involved

| | |
|---|---|
| *cfroot*/wwwroot/com/delmore/jxl.cfc | CFC that calls JXL API |
| *cfroot*/CustomTags/jxl.cfm | Custom Tag – CF_JXL |
| *cfroot*/CustomTags/jxlparam.cfm | Custom Tag – CF_JXLPARAM |
| *cfroot*/wwwroot/examples/useJXLCFC.cfm | Template calling CFC |
| *cfroot*/wwwroot/examples/useJXLTAG.cfm | Template calling Tags |




## Integrating JSPs & Servlets

- The PageContext class is an abstract class designed to be extended and implemented for specific JSP engine runtime environments
- Provides facilities to:
  - Write output to the client
  - Manage scoped namespaces
  - Manage session usage by page
  - Redirect requests to another resource
  - Include other pages within the same request
  - Handle exception processing

## Integrating JSPs & Servlets

- Demonstration
  - List the members of the PageContext class - DumpPageContext.cfm

```
<!--- List the members of PageContext --->
<cfdump var="#getPageContext()#" />
```




## Integrating JSPs & Servlets

- Demonstration
  - List the HTTP headers for the current request - GetHeaders.cfm

```
<!--- Get the ServletRequest from PageContext --->
<cfset servletRequest = getPageContext().getRequest()/>

<!--- Get an Enumeration of Header names --->
<cfset names = servletRequest.getHeaderNames()/>
<p>Request Headers</p>

<cfloop condition="#names.hasMoreElements()#">
   <cfset next = names.nextElement() />
   <cfoutput>
      <b>#next#</b> - #servletRequest.getHeader(next)#
   </cfoutput>
</cfloop>
```




## Integrating JSPs & Servlets

- ColdFusion pages, JSP pages and Servlets can interoperate in several ways
  - ColdFusion pages can invoke JSP pages and Servlets
  - JSP pages and Servlets can invoke ColdFusion pages
  - ColdFusion pages, JSP pages and Servlets share data in the Request, Session, and Application scopes (Note: Shared Request scope variable names in JSP pages or servlets must be all-lowercase.)




## Integrating JSPs & Servlets

- Demonstration
  - Accessing a JSP page from a ColdFusion page - JSPInclude.cfm

```
<!--- Include the Header --->
<cfset getPageContext().include("Header.jsp") />

<!--- Include Body for content --->
<cfinclude template="Body.cfm"/>

<!--- Include the Footer --->
<cfset getPageContext().include("Footer.jsp")/>
```




## Integrating JSPs & Servlets

- To access a servlet that exposes the same functionality, you use the same code.
- For example, to run a servlet called HelloWorldServlet
  - put the servlet class file in the serverroot/WEB-INF/classes directory
  - refer to the servlet with the URL /servlet/HelloWorldServlet

```
<!--- Include the servlet --->
<cfset getPageContext().include("/servlet/HelloWorldServlet")/>
```

  - *Note: You do NOT have to modify web.xml for this to work.*




## Integrating JSPs & Servlets

- Demonstration
  - Accessing a ColdFusion page from a JSP page - CallingCF.jsp

```
<jsp:include page="Header.cfm">
   <jsp:param name="Title" value="ColdFusion-Java Hybrid"/>
</jsp:include>
<!--- Get the body Content from Header.cfm file --->
<p><%= request.getAttribute("bodycontent") %></p>
<!--- Include the Footer --->
<%@ include file="Footer.jsp" %>
```

```
<!--- Header.cfm --->
<cfsetting showdebugoutput="no"/>
<html>
<head><title><cfoutput>#url.title#</cfoutput></title></head>
<body>
The title is <cfoutput>#url.title#</cfoutput>
<cfset request.bodyContent = "Here is the body content."/>
```

## Using Custom JSP Tags

- Custom JSP tags allow functionality to be boxed and reused across applications
- Custom JSP tags:
  - Can be customized via attributes from the calling page
  - Have access to shared scopes
  - Can modify the response generated by the calling page
  - Can communicate with each other (Create a variable that refers to a bean in one tag and then use the same bean in another tag)
  - Can be nested within one another
  - **Are a lot like ColdFusion tags**!




## Using Custom JSP Tags

- To use a custom tag
  - Put the tag library, consisting of the *taglibname*.jar file, and the *taglibname*.tld file, if one is supplied, in the *web_root*/WEB-INF/lib directory.
  - The JSP custom tag library **must be in this directory** for you to use the cfimport tag.
  - Restart ColdFusion.
  - In the ColdFusion page that uses a JSP tag from the tag library, specify the tag library name in a **cfimport** tag; for example: <cfimport taglib="/WEB-INF/lib/random.jar" prefix="random">
  - **Note:** The cfimport tag must be on the page that uses the imported tag. You cannot put the cfimport tag in Application.cfm.
  - Use the custom tag using the form <*prefix*:*tagName*>




## Using Custom JSP Tags

- Example - Jakarta String Tag Library
  - Importing and using a JSP Tag Library in ColdFusion
  - First, put taglibs-string.tld, taglibs-string.jar, and commons-lang-2.1.jar in WEB-INF/lib

```
<!--- JSPTagLib.cfm--->
<!--- Import the tag library --->
<cfimport taglib="/WEB-INF/lib/taglibs-string.jar" prefix="string">

<string:capitalize>capitalize the first letter of each word for
   me</string:capitalize><br/>
<string:split var="splitString" separator=" ">split my
   string</string:split><br/>
<cfdump var="#splitString#"/><br/>
<string:split var="splitString" separator=" ">split my
   string</string:split>
<string:wordWrap width ="20" delimiter="<br/>">Word-wrap a String. This
   involves formatting a string to fit in character width of
   page.</string:wordWrap>
```




## Using Custom JSP Tags

- Example - Jakarta String Tag Library - JSPTagLib.cfm
  - Output from example

Capitalize My String For Me

| array |
| --- |
| 1 split |
| 2 my |
| 3 string |

Word-wrap a String. This involves formatting a string to fit in character width of page.




## Using Custom JSP Tags

- Why use custom JSP tags vs. ColdFusion custom tags?
  - Leverage existing expanse of publicly available JSP tag libraries (Jakarta Taglibs Project and others)
  - Leverage JSP tag libraries already developed within your organization
  - Interoperate with existing J2EE systems within your organization that implement JSP tag libraries




## Calling ColdFusion from Java classes?

- Starting with ColdFusion 7.0.1, you can call ColdFusion Components (CFCs) from Java classes using the CFCProxy API.
- To call a CFC, the class must be in the ColdFusion Classpath (set in the Administrator/jvm.config)
- To compile, you will need cfusion.jar in your classpath

```
javac.exe -classpath C:\CFusionMX7\lib\cfusion.jar
CFCInvoker.java
```

## Calling ColdFusion from Java classes?

- Demonstration
  - Calling a CFC from a Java class
  - Files Involved

| | |
|---|---|
| *cfroot*/wwwroot/com/delmore/CFCInvoker.java | Java source file |
| *cfroot*/wwwroot/com/delmore/CFCInvoker.class | Java class file |
| *cfroot*/wwwroot/com/delmore/dataholder.cfc | CFC that returns a struct |
| *cfroot*/wwwroot/invoker.cfm | Template that instantiates CFCInvoker |




## Calling ColdFusion from Java classes?

- Caveats
  - This is not a well documented feature
  - It's for advanced users only
  - There are some situations where this can be very useful but in general there are less complex ways to achieve the same goal.
  - Beware the configuration… the java class must be in the ColdFusion classpath and you must compile with the same JVM version the server is running.




## Limitations of Java in ColdFusion

- There are some limitations of Java in ColdFusion
  - Object construction – init() method conflicts
  - Overloaded methods, resolving ambiguity with JavaCast()
  - ColdFusion case insensitivity
    - For example, JSPs must use all lowercase characters to refer to all request scope variables shared with CFML pages. You can use any case on the CFML page, but if you use mixed case to all uppercase on the JSP page, the variable will not get its value ColdFusion page.
  - Changes to compiled Java classes or .jar files require restart




## Putting It All Together

- What we already knew
  - ColdFusion is the tool of choice for developing feature rich applications quickly and easily!
  - Java provides a solid set of tools for developing powerful, enterprise-scale applications
- What we've learned
  - ColdFusion provides many ways of leveraging Java and related technologies!
  - Hybrid applications are powerful! They leverage the productivity and richness of ColdFusion as well as the strength and ubiquity of Java

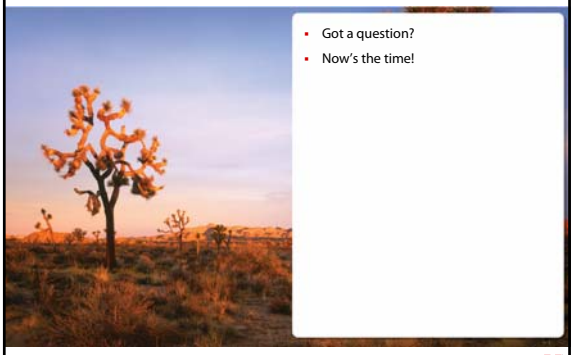



## Some Suggestions

- Keep it Easy! Abstract the Java!
  - Use CFCs as wrappers to your Java API calls
  - If you want to use your functionality as a custom tag, add a CFM that passes on the tag attributes to a function in your CFC
- Use createObject()
- Create an init function in your CFCs, have it return *this*
- Set a coding standard for variable names that includes how to handle case
- Use Application.cfc




## Questions and Answers



- Got a question?
- Now's the time!

## Where to Learn More

- **ColdFusion Developer Center – Java and J2EE Architecture**
  http://www.adobe.com/devnet/coldfusion/java_j2ee.html
- **LiveDocs – Integrating J2EE and Java Elements in CFML Applications**
  http://livedocs.macromedia.com/coldfusion/7/htmldocs/00001557.htm
- **Reality ColdFusion: J2EE Integration**
  http://www.forta.com/books/0321129482/




Better by Adobe™