

Building Hybrid Applications with ColdFusion and Java

Jason J Delmore

Senior Product Manager, ColdFusion

Adobe Systems, Inc.

jdelmore@adobe.com



Biography

Jason J Delmore

Senior Product Manager, ColdFusion
Adobe Systems Inc.

jdelmore@adobe.com

<http://www.cfinsider.com/>

- Began developing ColdFusion applications in 1999.
- Architected and led the development of several enterprise level applications.
- Very experienced in leveraging Java from within ColdFusion.
- NOT a java developer...

Agenda

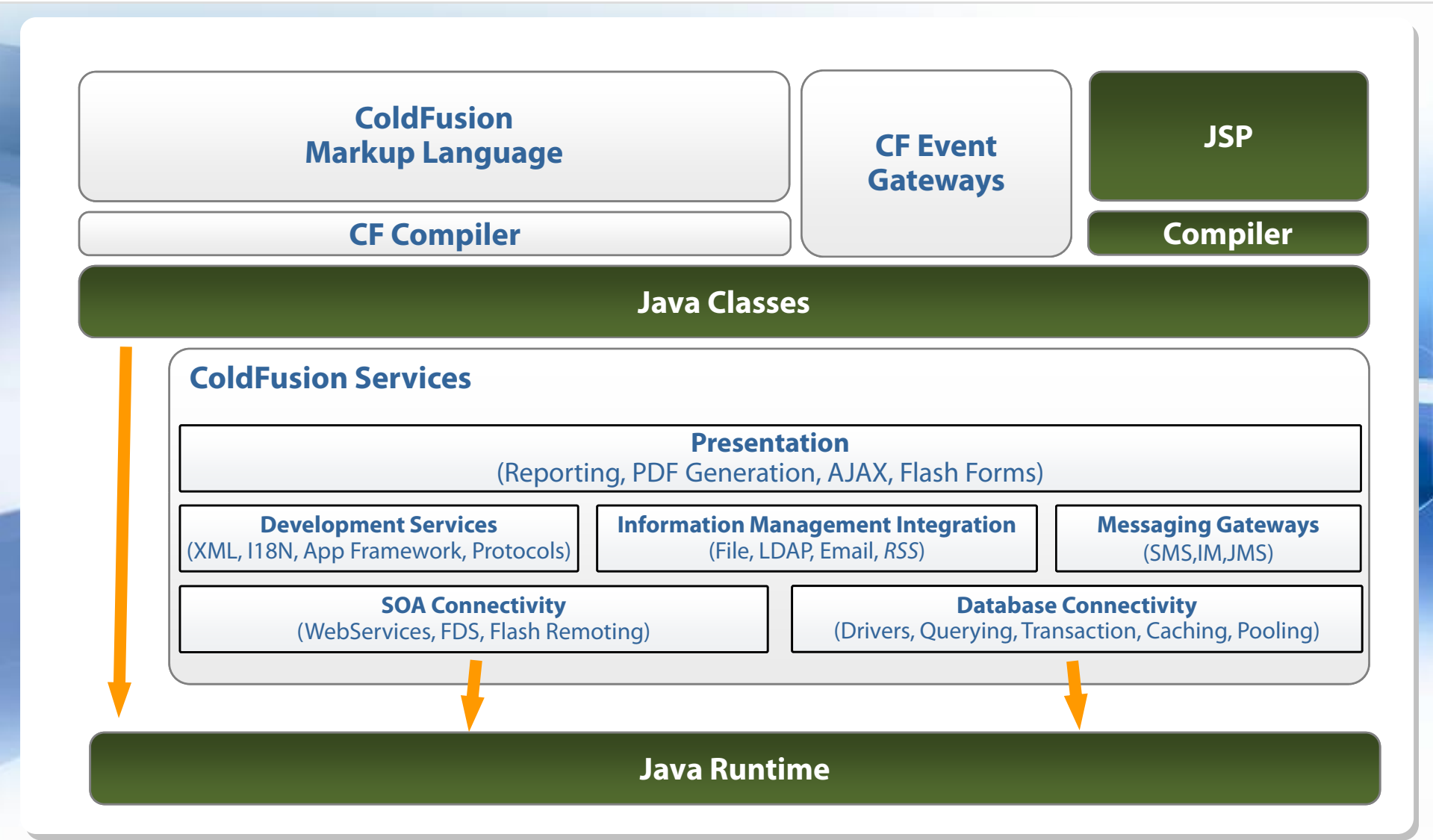
- What is ColdFusion?
- ColdFusion and Java
 - Why use them together?
 - How do I do it?
 - Limitations
 - Some Suggestions
- Questions and Answers
- Where to Find Out More

So, what *is* ColdFusion anyway?

Adobe® ColdFusion® is the fastest and easiest way to build **Java based** applications and web sites.

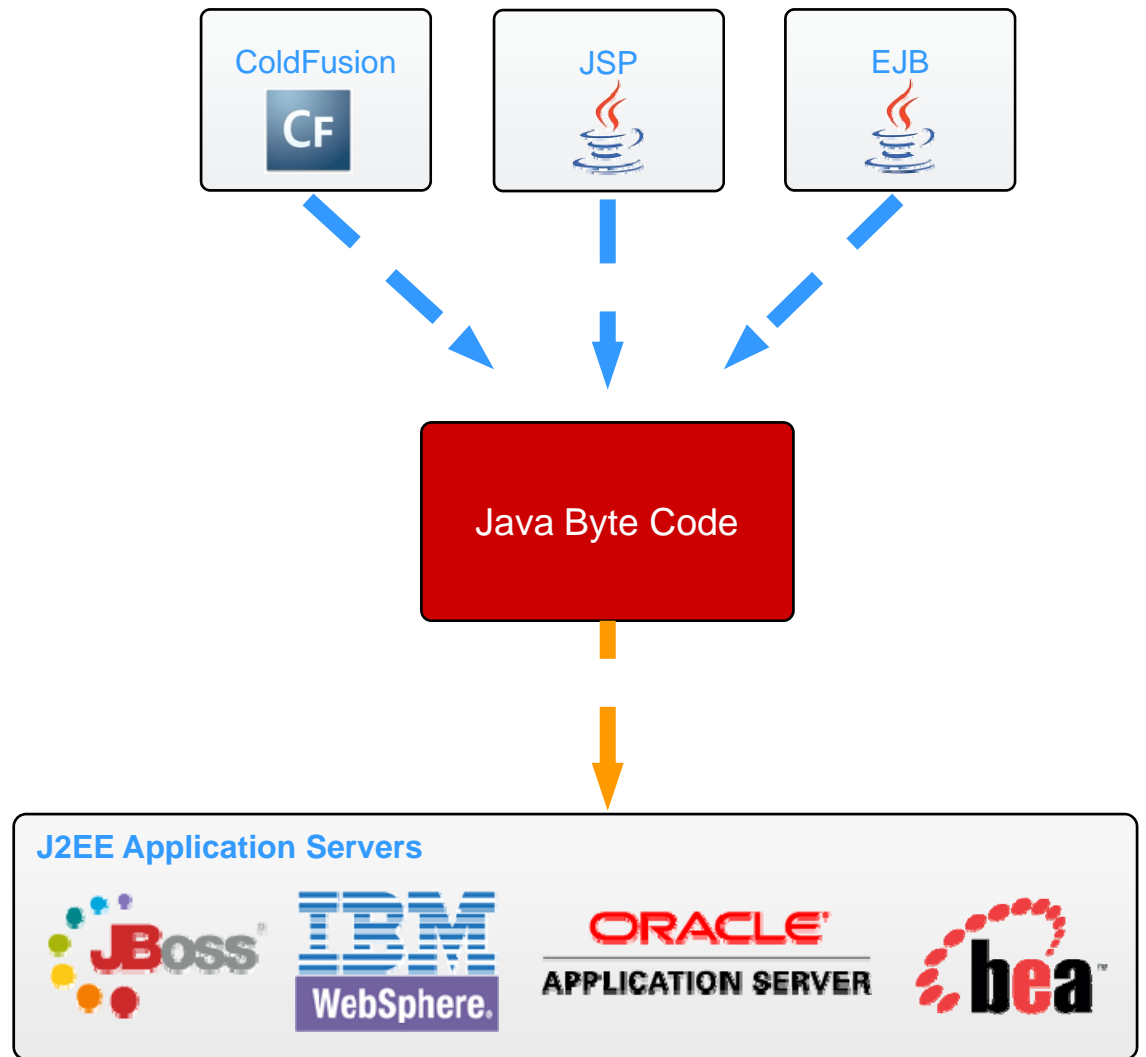
- ColdFusion **is** a J2EE Certified Java Application
- ColdFusion *provides* two languages, a web application server, a report building tool, an administrative tool, a server monitoring tool, a set of services for productivity, a library of java applications and...
- ColdFusion is a **Java-based Rapid Application Development** environment

Product Architecture



ColdFusion is Java... and more!

- Productivity
- Lower Development Costs
- Lower Maintenance Costs
- Services and features not natively available to Java/.NET



Why use ColdFusion and Java Together?

- **The problem**

- ColdFusion doesn't provide everything out of the box
- Java applications are typically complex and take longer to develop

- **The solution**

- ColdFusion provides quick and easy application development
- Java provides more "stuff"
- Hybrid applications leverage the ease of ColdFusion and the ubiquity of Java

How do I do it?

- Leveraging Java from CF
 - Java CFX Tags
 - JSP
 - Servlets
 - JSP Tag Libraries
 - Direct Invocation
- Can I leverage CF from Java? ~~Yes!!!~~ Hmm... sort of.
 - JSP
 - CFC Proxy
 - Web Services

Java CFX Tags

- Available since **ColdFusion 4.5**
- Java CFX API (cfx.jar) is still around
- Allowed developers to extend the functionality of ColdFusion pages beyond CFML

```
// 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 ) ;
    }
}
```

Java CFX Tags

- Limitations of Java CFXs (there are many!)
 - They must be registered within the ColdFusion Administrator
 - Poor datatype support - only strings and queries
 - No built-in exception handling

WWJD?: Don't use Java CFX Tags. Ever.

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:
 - **Include other pages within the same request**
 - **Write output to the client**
 - Manage scoped namespaces
 - Manage session usage by page
 - Redirect requests to another resource
 - Handle exception processing

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

- Accessing a ColdFusion page from a JSP page - [CallingCF.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."/>
```

```
<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" %>
```

Using JSP Tag Libraries

■ 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 JSP Tag Libraries

■ 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 JSP Tag Libraries

- 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 JSP Tag Libraries

- Why use JSP Tag Libraries?
 - Flex Tag Library (Web-tier compiler... built-in if you check the LCDS)
 - Jakarta Taglibs Project
 - Strings, l18N, Cache, Regexp, Scrape, and more...
 - Google Tag Library
 - JCE TagLib (Cryptography)
 - JPA TagLib (Persistence)
 - Leverage JSP tag libraries already developed within your organization

Direct Invocation

- **<CFOBJECT>**

- Tag to create objects... CFCs, Java, COM, Corba, WebService

- **<CFINVOKE>**

- Tag to invoke **methods** – CFINVOKE **does not create objects**
- **Performance pitfall:** If you don't 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

WWJD?: Always use CreateObject()! You really can't lose.

Direct Invocation

- **Everything** in ColdFusion is a Java Object!!!

Simple Types

a="1";

b=1;

c=true;

d=0+1;

isBoolean

YES

Java Object Type

java.lang.String

java.lang.Double

Direct Invocation

- **Everything** in ColdFusion is a Java Object!!!

Complex Types

ColdFusion

Java Object Type

```
e=["1"];
```

Array

`coldfusion.runtime.Array`
extends `java.util.Vector`

```
g={};
```

Struct

`coldfusion.runtime.Struct`
extends `java.util.FastHashTable`

Direct Invocation

- Which one of these is not like the others?
 - `a = Structnew();`
 - `b = {};`
 - `c = CreateObject("java", "coldfusion.runtime.Struct").init();`

Answer: C – because its blue

Direct Invocation

- **J2SE & J2EE APIs** are already in ColdFusion!!!
 - Static members are always available
 - Instance members available after instantiating the object
 - Call `init()` to instantiate
 - ColdFusion will give it an honest shot to implicitly instantiate the object if you don't do it (won't work on interfaces or abstract classes)

Working with Java Objects

- Okay... so the stuff is java objects... Now what? What can I do with a `java.lang.String`???

`charAt(int index)`

Returns the char value at the specified index.

`codePointAt(int index)`

Returns the character (Unicode code point) at the specified index.

`codePointBefore(int index)`

Returns the character (Unicode code point) before the specified index.

`codePointCount(int beginIndex, int endIndex)`

Returns the number of Unicode code points in the specified text range of this `String`.

`compareTo(String anotherString)`

Compares two strings lexicographically.

`compareToIgnoreCase(String str)`

Compares two strings lexicographically, ignoring case differences.

`concat(String str)`

Concatenates the specified string to the end of this string.

`contains(CharSequence s)`

Returns true if and only if this string contains the specified sequence of char values.

`contentEquals(CharSequence cs)`

Returns true if and only if this `String` represents the same sequence of char values as the specified sequence.

`contentEquals(StringBuffer sb)`

Returns true if and only if this `String` represents the same sequence of characters as the specified `StringBuffer`.

`copyValueOf(char[] data)`

Returns a `String` that represents the character sequence in the array specified.

`copyValueOf(char[] data, int offset, int count)`

Returns a `String` that represents the character sequence in the array specified.

`endsWith(String suffix)`

Tests if this string ends with the specified suffix.

`equals(Object anObject)`

Compares this string to the specified object.

`equalsIgnoreCase(String anotherString)`

Compares this `String` to another `String`, ignoring case considerations.

Working with Java Objects

- Example – Working with Strings

```
<cfset myString = "Hello World"/>
<cfdump var="#myString.length()" /><br />
<cfdump var="#myString.toLowerCase()" /><br />
<cfdump var="#myString.toUpperCase()" /><br />
<cfdump var="#myString.replace('World', 'Everyone')#" /><br />
<cfdump var="#myString#" /><br />
```

```
11
hello world
HELLO WORLD
Hello Everyone
Hello World
```

Working with Java Objects

- 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("ColdFusion")/>
<!-- append to the buffer -->
<cfset StringBuffer.append("Rocks!")/>
<!-- insert into the middle of the buffer -->
<cfset StringBuffer.append(" 8 ")/>
<!-- output the buffer -->
<cfoutput>#StringBuffer.toString()#</cfoutput>
<!-- reverse the buffer -->
<cfset StringBuffer.reverse()/>
<cfoutput>#StringBuffer.toString()#</cfoutput>
```

Output

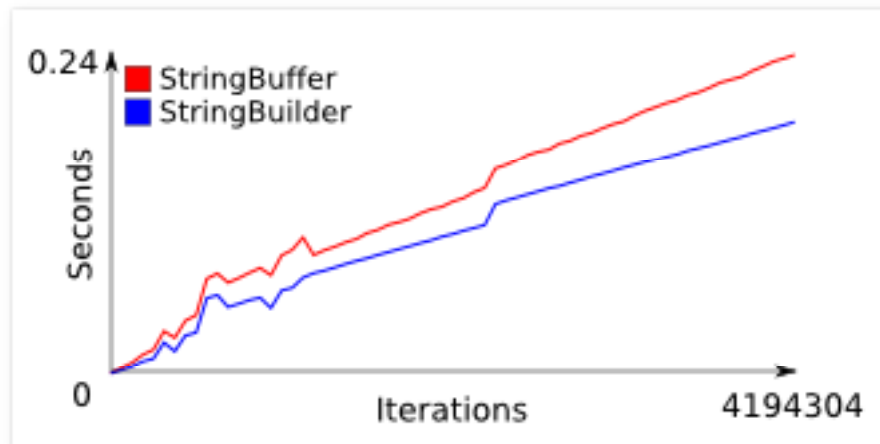
ColdFusion 8 Rocks!
!skcoR 8 noisuFdloC

Working with Java Objects

- Why use a StringBuffer???
- Performance
- StringBuffers are safe for use by multiple-threads
 - Automatically synchronized
- **Mutability**... “Strings” are immutable (once constructed they can’t be changed) when you call a function on a StringBuffer, it actually changes the object

Working with Java Objects

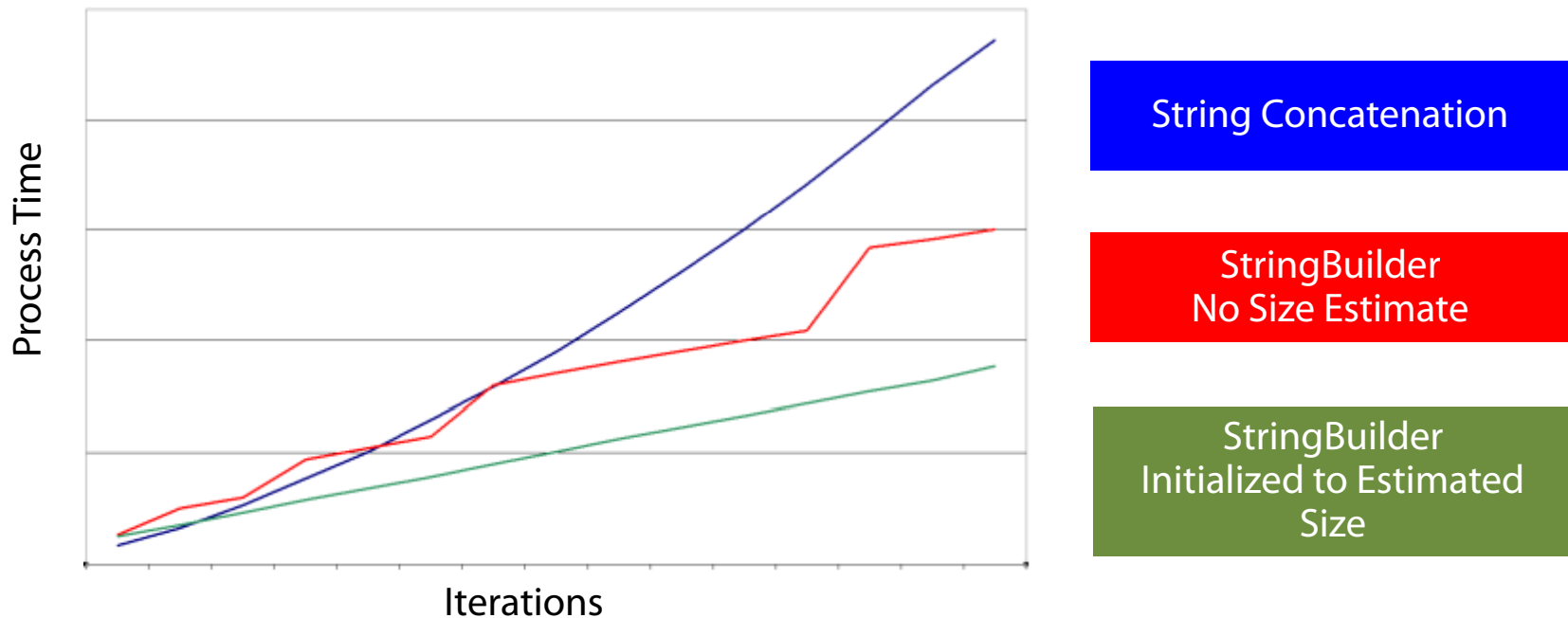
- Also... JDK 1.5 Introduced StringBuilder
 - The same as StringBuffer
 - But faster
 - Only for use when **single threaded**



StringBuilder vs StringBuffer

Working with Java Objects

- StringBuilder versus Strings for Concatenation



WWJD?: Use a StringBuilder/StringBuffer if you're concatenating strings in hundreds/thousands of iterations.

Calling Java Objects

- Demonstration – ColdFiSH

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

- **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* and *cfroot/gateway/lib*
 - The J2EE container will load
 - *cfroot/WEB-INF/lib* and *cfroot/WEB-INF/classes*

■ 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`

WWJD?: Add your own directory to the CF Classpath

Calling Java Objects

- There are a lot of Java APIs included already
 - ANT Java Build Tool
 - iText PDF Manipulation
 - POI M\$FT File Manipulation
 - Log4J Logging Utilities
 - AXIS Web Services
 - JAI Java Advanced Imaging

Calling Java Objects

- There are a lot of Java APIs included already
- Apache Commons
 - **BeanUtils** Java reflection and introspection
 - **Codec** General encoding/decoding algorithms (for example phonetic, base64, URL).
 - **Collections** Extends or augments the Java Collections Framework.
 - **Digester** XML-to-Java-object mapping utility.
 - **Discovery** Tools for locating resources by mapping service/reference names to resource names.
 - **Net** Network utilities and protocol implementations.

Calling Java Objects

- Example

- Rotating a PDF with iText

itext.cfm

Calling Java Objects

■ Demonstration

- Leveraging JExcel API to build Excel Documents in ColdFusion

- Files Involved

- | | |
|---|--------------------------|
| ■ <i>cfroot/com/cfinsider/jxl.cfc</i> | CFC that calls JXL API |
| ■ <i>cfroot/CustomTags/jxl.cfm</i> | Custom Tag – CF_JXL |
| ■ <i>cfroot/CustomTags/jxlparam.cfm</i> | Custom Tag – CF_JXLPARAM |
| ■ <i>cfroot/jxlcfc/examples/useJXLCFC.cfm</i> | Template calling CFC |
| ■ <i>cfroot/jxlcfc/examples/useJXLTAG.cfm</i> | Template calling Tags |

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?

- 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

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
- 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`

What Would Jason Do?

- Don't use Java CFX, ever.
- Always use `createObject()`
- Use a `StringBuilder/StringBuffer` if you're concatenating strings in hundreds/thousands of iterations.
- Add your own directory to the CF `ClassPath`

Putting It All Together

- **What we already knew**

- ColdFusion is the tool of choice for developing feature rich applications quickly and easily!
- Java has a rich library of interesting APIs (some built-in but most from the Open Source community)

- **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

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.adobe.com/coldfusion/8/htmldocs/help.html?content=Java_1.html
- **Reality ColdFusion: J2EE Integration**
<http://www.forta.com/books/0321129482/>

Better by Adobe.™