Leveraging Java from CFML with Groovy

- Barney Boisvert
- Sr. Web Application Developer at Mentor Graphics
- CF Developer since 1999

Leveraging Java from CFML with Groovy

- CFML is great
- CFML isn't so hot
- Java is great
- Java isn't so hot
- CFML + Java is great
- CFML + Java isn't so hot
- CFML + Groovy + Java is great
- CFML + Groovy + Java is great

The Basics

- What is Java?
- What is a JVM?
- What is Groovy (or ColdFusion, JRuby, etc.)?

The CFML Lifecycle

- CFML source (*.cfm/*.cfc)
- Server JIT compiles/recompiles CFML
- Bytecode in memory
- [native JIT into machine code]

The Java Lifecycle

- Java source (*.java)
- Javac compiles Java
- Bytecode saved to disk
- Bytecode loaded into JVM
- Bytecode in memory
- [maybe JIT into machine code]

The Groovy Lifecycle

- Can use either CFML or Java lifecycle
  - Dynamic source load by destination JVM
  - Bytecode compilation for loading into future JVM

The Point: It's all Bytecode

- CFML ends up as bytecode
- Java ends up as bytecode
- Groovy ends up as bytecode

By the time it runs, it's all the same

Why Use Java

- Rich collections framework
- Rich class libraries
- Low-level access

Why NOT Use Java

- CFML/Java object mismatch
- Ugly CFML syntax for Java
- New development workflow
  - Compilation
  - Container restarts
- Data type conversion issues
- “scary”

How Can Groovy Help?

- It matches Java symantics
- It matches Java syntax (99%)
- It can deal with CF data structures
- It can be dynamically loaded

A Comparison

- Sort an array of strings
- CFML: `arraySort(myArray, “text”)`
- Java: `createObject(“java”, “java.util.Collections”).sort(myArray)`
- Groovy: `Collections.sort(myArray)`

Another Comparison

- Sort an array of structs (by the value of the 'letter' key)
- CFML

```
<cfset keys = "" />
<cfloop from="1" to="#arrayLen(myArray)#" index="i">
    <cfset keys = listAppend(keys, myArray[i].letter & "~" & i) />
</cfloop>
<cfset newArray = [] />
<cfloop list="#listSort(keys, 'textNoCase')#" index="key">
    <cfset arrayAppend(newArray, myArray[listLast(key, "~")]) />
</cfloop>
```

Another Comparison

- Java

```
import java.util.Comparator;
import java.util.Map;
public class LetterKeyComparator implements Comparator {
    public int compare(Object o1, Object o2) {
        return ((Comparable)((Map)o1).get(“letter”)).compareTo(
            ((Map)o2).get(“letter”)
        );
    }
}
```

Compile, put .class in /WEB-INF/classes, restart container

```
createObject("java", "java.util.Collections").sort(
    myArray,
    new LetterKeyComparator()
)
```

Another Comparison

- Groovy (Java analogue)

```
class LetterKeyComparator implements Comparator {
    int compare(o1, o2) {
        o1.letter.compareTo(o2.letter)
    }
}

Collections.sort(myArray, new LetterKeyComparator())
```

- Groovy (the easy way)

```
Collections.sort(myArray, {o1, o2 ->
    o1.letter.compareTo(o2.letter)
} as Comparator)
```

Closure Syntax

```
//Groovy
{o1, o2 ->
    o1.letter.compareTo(o2.letter)
}
```

```
//JavaScript
function(o1, o2) {
    return o1.letter.compareTo(o2.letter);
}
```

- Arguments are optional
  - Single argument named 'it'
- No 'function' keyword
- Optional 'return' keyword

Neat. How to Use?

- Download the Groovy JAR
- Drop in /WEB-INF/lib
- Restart container
- Load a ScriptEngine
- Configure a Binding
- Create a script
- Hand to ScriptEngine w/ Binding
- Execute

The Real Way

- Download CFGroovy
- Install groovy-all-x.y.z.jar
- Use `<g:script>` just like `<cfscript>`

```
<cfimport prefix="g" taglib="groovyEngine/tags" />

<cfset myArray = listToArray("barney,heather,lindsay") />

<cfdump var="#myArray#" />

<g:script>
    variables.myArray.add("emery")
</g:script>

<cfdump var="#myArray#" />
```

Where's the Code??

- In Eclipse, naturally.
- All examples on Railo 3 RC1 (that's 3.0.0.003)
- CFGroovy works on all Railo 3 and CF8.0.1

I'm Sleepy

- CF Groovy includes Hibernate
- Only persists Groovy entities, not CFCs
- No Hibernate XML (unless you want to)
- Autoreloading, compile-on-the-fly

Hibernate

- “The” Java ORM framework
  - Runtime (bytecode) centric
  - State-based
  - HQL query language (SQL available)
  - “universal” DB support
  - Implements EJB
- Part of JBoss JEMS
- Works bottom-up or top-down
- Supports Java Annotations or XML config

A Groovy Entity

```
package com.barneyb

import javax.persistence.*

@Entity
class User extends AbstractEntity {
                                                                    ...
    String firstName
    String lastName
    String email

    @Column(unique = true)
    String username
}
```

A Groovy Entity

```groovy
package com.barneyb

import javax.persistence.*

@Entity
class User extends AbstractEntity {

    String firstName
    String lastName
    String email

    @Column(unique = true)
    String username
}
```

```groovy
package com.barneyb

import javax.persistence.*

@MappedSuperclass
class AbstractEntity {

    @Id
    @GeneratedValue
    Long id

    @Version
    Long version

    @Temporal(TemporalType.TIMESTAMP)
    Date createDate

    @Temporal(TemporalType.TIMESTAMP)
    Date modifyDate

}
```

Persist It!

```
<g:script>
import com.barneyb.*

variables.user = new User([
    firstName: attributes.firstName,
    lastName: attributes.lastName,
    username: attributes.username,
    email: attributes.email,
    createDate: new Date(),
    modifyDate: new Date()
])
request.sess.save(variables.user)
request.sess.flush()
</g:script>
```

Update It!

```
<g:script>
import com.barneyb.*

variables.user = request.sess.get(User.class, Long.parseLong(attributes.id))
variables.user.firstName = attributes.firstName
variables.user.lastName = attributes.lastName
variables.user.username = attributes.username
variables.user.email = attributes.email
variables.user.modifyDate = new Date()
request.sess.save(variables.user)
request.sess.flush()
</g:script>
```

Closing Thoughts

- Use version control. Always.
- Use frameworks: don't reinvent the wheel.

- bboisvert@gmail.com
- http://www.barneyb.com/