



Getting Dynamic with ColdFusion

Sean Corfield
Chief Systems Architect & VP Engineering
Broadchoice, Inc.

4th-6th June 2008
Edinburgh, Scotland

SCOTCH ON THE ROCKS

What is he on about?

- Java** developers are looking at JRuby and Groovy to get things built quicker
- CFML has more in common with Ruby and Groovy than with Java
- But you have to think dynamically to leverage those advantages
- ** Okay, **Some** Java developers are...

Who is this man?

- Chief Systems Architect / VP Engineering @ Broadchoice, Inc.
- Adobe Community Expert for ColdFusion
- Manager of Bay Area ColdFusion User Group
- Formerly Senior Architect for Macromedia IT
- ColdFusion developer since 2001 ("Neo")
- Previously Java developer (since 1997) and C++ developer (since 1992)

Agenda

- Dynamic Typing
- Behavior versus Type
- Dynamic Behavior
- Dynamic Code
- Downsides

Dynamic Typing

4th-6th June 2008
Edinburgh, Scotland

SCOTCH ON THE ROCKS

Dynamic Typing

- What is a type?

Dynamic Typing

- What is a type?
 - numeric, date, string, boolean etc

Dynamic Typing

- What is a type?
 - numeric, date, string, boolean etc

Dynamic Typing

- What is a type?
 - numeric, date, string, boolean etc
- What has a type?

Dynamic Typing

- What is a type?
 - numeric, date, string, boolean etc
- What has a type?
 - "123" - numeric? string? boolean?

Dynamic Typing

- What is a type?
 - numeric, date, string, boolean etc
- What has a type?
 - "123" - numeric? string? boolean?
 - yes, it's all of those

Dynamic Typing

- What is a type?
 - numeric, date, string, boolean etc
- What has a type?
 - "123" - numeric? string? boolean?
 - yes, it's all of those
 - `isBoolean("123") = YES`

Dynamic Typing (II)

4th-6th June 2008
Edinburgh, Scotland

SCOTCH ON THE ROCKS

Dynamic Typing (II)

- In Java, variables have type:

```
int x = 42;
```

```
x = "42"; // is not allowed
```

Dynamic Typing (II)

- In Java, variables have type:

```
int x = 42;
```

```
x = "42"; // is not allowed
```

- In CFML, **values** have type
 - a variable can contain **any** value
 - the actual value determines what operations are permitted on that expression

```
("12" & "34") / "2" is 617
```

Dynamic Typing (III)

4th-6th June 2008
Edinburgh, Scotland

SCOTCH ON THE ROCKS

Dynamic Typing (III)

- In Java, an object's type determines what operations are permitted on that object

Dynamic Typing (III)

- In Java, an object's type determines what operations are permitted on that object
- In CFML, an object's methods determine what operations are permitted
 - Specifically, which methods are available at **runtime** on a given object **instance**

Behavior versus Type

```
dave.study();
```

```
dave.takeTest();
```

```
dave.fly();
```

- This will run if dave is an object with all three methods - regardless of the actual type you might think **dave** has
- Demo of flying students >>>

Behavior versus Type (II)

- Two objects may have "similar" methods but be otherwise unrelated
 - If they can be treated identically by client code, their behavior is more important than their type
- Small demo and bigger example >>>

Behavior versus Type (III)

- In Java, an interface is required to allow two otherwise unrelated classes to be treated as the same "type"
- Dynamic languages don't need interfaces (Ruby and Smalltalk do not have interfaces)
- Design by Contract means programming to an interface (small 'i')

Design by Contract

- I tell you (somehow) what I expect from you
- You satisfy that contract
- I do the right thing

DbC in Java

- Java interfaces can be a contract but they are not flexible - not dynamic

```
interface IContract { ... }
```

```
class X implements IContract { ... }
```

- To use a third-party class, you may need to create a new class that extends the third-party class and implements the contract
- Then you must use this new "artificial" class

DbC in CFML

- Document the required behavior
 - I tell you what I expect from you
- Pass an object that provides that behavior
 - It satisfies the contract
- The code will do the right thing

- Even with `onMissingMethod()`
 - `<cfinterface>` doesn't like `onMissingMethod()`

Dynamic Typing (IV)

- Instead of using argument types and return types, we can say `returntype="any"` or `type="any"` and be completely dynamic
- Amongst other things, this provides a nice clean solution to the "CFML doesn't have null" problem

Handling Null in Java

```
if ( someCondition ) {  
    return null;  
} else {  
    return someObject;  
}
```

```
if ( result != null ) ...
```

Handling Null in CFML

```
if ( someCondition ) {  
    return 0;  
} else {  
    return someObject;  
}
```

```
if ( isObject(result) ) ...
```

Dynamic Behavior

- The behavior of an object is not fixed
- The methods can be changed at runtime
- Methods can be called, even if they don't exist, using `onMissingMethod()`
- Let's look at `onMissingMethod()`...

Dynamic Delegation

- Service to Gateway
- In simple applications, the Service layer is full of methods that just call the same method on the Data layer
- Use `onMissingMethod()` to provide direct invocation of same method on gateway
- Delegation example >>>

Dynamic Boilerplate

- Example:
 - `UserGateway.getUserById(id)`
 - `AccountGateway.getAccountById(id)`
- Even with an ORM, these are tedious to write in every single data gateway
- Use `onMissingMethod()` in a base class to implement these
- Boilerplate example >>>

Dynamic Code

- Method renaming / aliasing
- Method injection
- Mixins

Dynamic Adapters

- Your object has the wrong method names to be used with a third-party library
- Just add aliases:

```
myObject.someMethod = myObject.myMethod;
```
- Related: Closures and Edmund allow method name to be specified dynamically

Method Injection

- Popular frameworks use this technique:
 - Transfer
 - ColdMock / CFEasyMock
- Can either:
 - Inject existing methods (from another object)
 - Write methods to a file and include that (injecting all the methods in the file)

Method Injection (II)

- Add simple method:

```
someObj.getValue = myGetValue;
```

```
function myGetValue(name) {  
    return variables[name];  
}
```

```
val = someObj.getValue("data");
```

Method Injection (III)

- Add all the methods from one object into another:

```
structAppend (someObj , anotherObj) ;
```

- This only mixes in public methods!

Method Injection (III)

- You really need to mixin private methods as well since public methods often rely on private helpers
- Dynamic mixin demo >>>

Downsides

- Without "compile-time" type information, it **is** easier to make mistakes and it **is** harder to find those bugs
- Unit testing becomes very, very important
 - This is why it's built into Rails, Grails etc
 - Learn to love MXUnit
 - Sometimes dynamic code can be easier to build tests for (especially mock objects)

Downsides (II)

- Step-thru debugging can be harder with dynamic code (since there's no source code available to work with at times)
- Extensive logging becomes important
 - Can be added dynamically via AOP so it doesn't have to be written into everything!

Downsides (III)

- Performance is lower
 - "Equivalent" code in CFML is slower than Java (mostly because it is not **equivalent**)
 - If you have a performance problem and your profiling shows that it really is the code
 - Swap out some dynamic code for "static" code, e.g., hand-code methods that you're implementing dynamically with `onMissingMethod()`
 - Use Java - if you care about performance that much

Summary

- ColdFusion != Java
 - Learn to think dynamically
- Common frameworks already work this way
 - ColdSpring, Transfer build code on the fly
- Leveraging dynamic code can help you build large systems quicker by writing less code... sometimes **much less code**
 - Delegation, boilerplate, cross-cutting concerns

Resources

- Design Patterns (2007)
 - <http://corfield.org/articles/DesignPatterns.pdf>
- Duck Typing (2008)
 - <http://corfield.org/articles/DuckTyping2008.pdf>

Q & A

- Sean Corfield
- sean@corfield.org
- <http://corfield.org/>
- scorfield@broadchoice.com
- <http://broadchoice.com/>