Learn You a What for Great Good?

Sean Corfield
World Singles llc

Polyglot Lessons to Improve Your CFML!

Sean Corfield
World Singles llc

Agenda

- Idioms in other languages
- Applying those in CFML
- Collection classes
- Arrays & Structs
- Closures
- Integrating other languages

You Might Prefer...

- Practical Deployment with Git and Ant
- Rich Apps with AngularJS
- Automating PhoneGap Build
- Writing Secure CFML

Me

- Functional Programming in the 80's
- Object-Oriented Programming in the 90's
- Web / Dynamic Programming in the 00's
- Mostly Clojure today

Me & Polyglot

- I love learning new programming languages!
- Learned a dozen languages at university
- Probably learned another dozen since
- Production apps in a dozen languages

Agenda

- **Idioms in other languages**
- **Applying those in CFML**
- Collection classes
- Arrays & Structs
- Closures
- Integrating other languages

JavaScript

- It's ubiquitous - see js.Objective()
- "OO" Prototype-based
- Heavy use of callbacks & closures

Prototypes

- var Person = function(first, last){
  this.first = first;
  this.last = last;
};
Person.prototype.greeting =
function(salutation){
  return salutation + " " + this.first + "!";
};

Prototypes (cont)

- var me = new Person("Sean", "Corfield");
  me.greeting("Hi"); // Hi Sean!
  Person.prototype.fullname = function(){
    return this.first + " " + this.last;
  };
  me.fullname(); // Sean Corfield
- Changes affect all live instances!

Callbacks

- Callbacks are a way for a process to initiate additional operations after it completes
- A process is passed a function that is calls when it has finished doing its job
- Most non-trivial JavaScript uses callbacks
- jQuery Ajax calls are asynch then the callback is invoked to handle the results

Closures

- A closure is a function expression that "closes over" part of its definition environment
- WAT?

Closures

- A function expression that uses variables that were in scope when it was defined
- Just show me an example!

Closures

- var greeting = function(salutation){
  return function(name){
  return salutation + " " + name + "!";
  };
};
var greet = greeting("Hello");
greet("Sean"); // Hello Sean!

JavaScript and CFML

- Prototypes have no CFML equivalent
- Can modify CFC metadata
  - singleton, per-class data / functions
  - doesn't affect existing instances
  - limited effect on new instances

JavaScript and CFML

- Can imitate with onMissingMethod
- and a public prototype member
- DEMO!

JavaScript and CFML

- Callbacks have been possible for ages
- CFML allows functions to be passed as args

JavaScript and CFML

- We don't do much async stuff in CFML tho'
- HTTP requests require synch results
  - so we tend to join threads
- Otherwise we just "fire'n'forget" threads

JavaScript and CFML

- As of ColdFusion 10 / Railo 4: closures!
- DEMO!

Groovy

- Designed to be a "better Java"
- Low ceremony
- Dynamic typing

Code Blocks

- { n -> n * n}
  { it * it }
  [1, 2, 3, 4].collect { it * it }
  [1, 2, 3, 4].findAll { it > 2 }
  [1, 2, 3, 4].each { println it }

Code Blocks

- { n -> n * n}
  { it * it }
  [1, 2, 3, 4].collect { it * it } // [1, 4, 9, 16]
  [1, 2, 3, 4].findAll { it > 2 } // [3, 4]
  [1, 2, 3, 4].each { println it }
  1
  2
  3
  4

Dynamic Typing

- String i = 42;
  Integer j = "42";
  // errors at RUNTIME, not compile time

Dynamic Typing

- def i = 42;
  def j = "42";
  // types are optional anyway; this is valid

Dynamic Typing

- Optional static typing is available in Groovy from 2.0 onwards...

Groovy and CFML

- function(n) { return n * n; } // { n -> n * n }
  function(it) { return it * it; } // { it * it }
- collect / findAll - we'll cover later
- Optional / dynamic typing is familiar, yes?

Clojure

- Lisp on the JVM
- Based on a number of abstractions
- General purpose replacement for Java
- Immutable data by default

Sequence Abstraction

- first, rest, cons
- may be countable (knows own length)
- map, filter, reduce
  - Groovy's collect, findAll - sort of

Sequence Abstraction

- (first [1 2 3 4])
  (rest [1 2 3 4])
  (cons 5 [6 7 8])
  (map inc [1 2 3 4])
  (filter even? [1 2 3 4])
  (reduce + [1 2 3 4])

Sequence Abstraction

- (first [1 2 3 4]) ;; 1
  (rest [1 2 3 4]) ;; (2 3 4) - not a vector
  (cons 5 [6 7 8]) ;; (5 6 7 8) - not a vector
  (map inc [1 2 3 4]) ;; (2 3 4 5)
  (filter even? [1 2 3 4]) ;; (2 4)
  (reduce + [1 2 3 4]) ;; 10

Lazy Sequences

- Can model infinite sequences
- Sequence realized on-demand
- Can wrap discrete processes to produce continuous processes-as-sequences

Lazy Sequence

- (iterate inc 0) ;; (0 1 2 3 4 5 6 7 8 9 10 11...
  (take 5 (iterate inc 0))
  (take 5 (drop 5 (iterate (fn [n] (* 2 n)) 1)))

  (take 5 (map inc (filter odd? (iterate inc 0))))

Lazy Sequence

- (iterate inc 0) ;; (0 1 2 3 4 5 6 7 8 9 10 11...
  (take 5 (iterate inc 0)) ;; (0 1 2 3 4)
  (take 5 (drop 5 (iterate (fn [n] (* 2 n)) 1)))
  ;; (32 64 128 256 512)
  (take 5 (map inc (filter odd? (iterate inc 0))))
  ;; (2 4 6 8 10)

Clojure and CFML

- No immutable data
  - can implement immutable objects
    - sort of - can often circumvent :(
- No sequence abstraction
  - just loops :(

Clojure and CFML

- Can imitate lazy sequences!
  - as a function that returns a pair
    - of the next value and a function for the rest of the lazy sequence
- DEMO!

Scala

- Designed to be a "better Java"
- Low ceremony
- Strong, static typing

OOP / FP Hybrid

- Full object-oriented language
- Classes + objects + case (value) objects
- Immutable data possible
- Sophisticated collection classes
- head, tail, + (cons), size
- map, filter, reduce, etc

Strong Typing w/ Inference

- Mostly can omit types (like dynamic langs)
- Still type-safe operations
- Also type-safe collections
  - List of integer can contain only integers

Scala and CFML

- Not much in common
- No strong typing, no immutable data, no real collection classes...
- Useful to "Think in FP vs OOP"

Agenda

- Idioms in other languages
- Applying those in CFML
- **Collection classes**
- Arrays & Structs
- Closures
- Integrating other languages

Collection Classes

- Lists, vectors, maps (structs), queues, stacks, sets, bags, ...
- Standard API (like Clojure)
- Standard functions to operate on them

Agenda

- Idioms in other languages
- Applying those in CFML
- Collection classes
- **Arrays & Structs**
- Closures
- Integrating other languages

Arrays & Structs

- Array can be treated as list or vector
- Struct is a map and can model a set
- ColdFusion 10 & Railo 4 introduced
  - arrayEach, arrayFilter, etc
  - structEach, structFilter, etc
  - no map or reduce functions :(

Agenda

- Idioms in other languages
- Applying those in CFML
- Collection classes
- Arrays & Structs
- **Closures**
- Integrating other languages

Closures

- ColdFusion 10 & Railo 4 introduced these
- Makes it easier to write map, reduce, etc
- Finally allows us to treat arrays and structs more like collection classes (in other langs)
- DEMO!

Agenda

- Idioms in other languages
- Applying those in CFML
- Collection classes
- Arrays & Structs
- Closures
- **Integrating other languages**

Integrating Other Languages

- Class path & libraries
  - JavaLoader helps but does not solve all problems, same for loadPaths (CF10)
- Compiling & JAR files

Integrating Other Languages

- Calling into other languages
  - Data structure interop
    - Some commonality in Java types
- Calling back into CFML
  - createDynamicProxy (CF10) can help

Summary

- Learn a new language every year?
- Better CFML through other languages
- Arrays and structs are very powerful (now)
  - use them as general collection types
  - closures make them much more powerful

Q & A?

- @seancorfield
- http://corfield.org
- sean@corfield.org