

# Adam Cameron's CFML Blog

TUESDAY, 25 FEBRUARY 2014

## ColdFusion 11: `.map()` and `.reduce()`

G'day:

More ColdFusion 11 testing. This time I look at the new `.map()` and `.reduce()` methods that each of array, struct and lists now have. It's mostly good news.

ColdFusion has increased its repertoire of object-iterator functions further in ColdFusion 11. In ColdFusion 10 it had the following:

- `listFilter()`
- `arrayEach()`
- `arrayFilter()`
- `structEach()`
- `structFilter()`

`listEach()`, for some reason was not implemented in ColdFusion 10, but is there in ColdFusion 11.

Just to recap on the `each()` and `filter()` functionality, here's a quick example of the array versions of each of them:

**`each()`**

```
letters = ["a","b","c","d"];
arrayEach(letters, function(){
    writeDump(arguments);
});
```

On ColdFusion 10, we get this:

struct	
1	a
struct	
1	b
struct	
1	c
struct	
1	d

So we see that `arrayEach()` does what it says on the tin: it calls the callback for each element of the array. And the callback receives the value of each array element. Now on ColdFusion 11, this code yields... bugger all. It doesn't error, but it doesn't do anything. This had me scratching my head for quite a while, but I just cracked it... On ColdFusion 11 one *must* specify the arguments of the callback (this should not be necessary), **thus**:

```
arrayEach(letters, function(v,i){
    writeDump(arguments);
});
```

Then it works:

struct	
I	1
V	a






### ABOUT

I've been a ColdFusion developer for over 10 years. At work or participating on various CF forums I spot a few "interesting" things: I'm going to discuss them here.

I tend to be a bit "forthright": I think CF is a good product, but I won't blindly defend it in all circumstances, and I have been quite critical of ColdFusion and Adobe at times. This will come out occasionally here: I make no apology for it.

Everything said here is my own opinion. Feel free to disagree with me :-)

### FEEDS / LINKS

-  [Default feed](#)
-  [Last ten articles \(full\)](#)
-  [Last ten articles \(summary\)](#)
-  [50-most recent ColdFusion Bugs](#)
-  [50-most recent ColdFusion Builder Bugs](#)
- [This blog's communication policy](#)
- [A better UI for searching the ColdFusion bugbase](#)

### LABELS

[ColdFusion](#) (181) [CFML](#) (174) [Adobe](#) (100) [Railo](#) (94) [Bugs](#) (82) [Community Members](#) (76) [ColdFusion 11](#) (59) [Code Examples](#) (53) [ColdFusion 10](#) (53) [Survey](#) (37) [Blog](#) (30) [Rhetoric](#) (30) [Courting Controversy](#) (28) [StackOverflow](#) (21) [OpenBD](#) (19) [Off Topic](#) (17) [Unit Testing](#) (17) [Conference](#) (16) [PHP](#) (16) [Theory](#) (16) [Application.cfc](#) (15) [CFCamp](#) (14) [CFlib](#) (14) [TDD](#) (14) [Regular expressions](#) (13) [Arrays](#) (12) [ColdFusion 9](#) (12) [Documentation](#) (12) [JSON](#) (12) [ColdFusion Builder](#) (11) [Ruby](#) (10) [Interfaces](#) (9) [CFClient](#) (8) [cf.Objective\(\)](#) (8) [CFHour](#) (7) [CFMLDeveloper](#) (7) [REST](#) (7) [Java](#) (6) [WebSockets](#) (6) [Beer](#) (5) [Coldbox](#) (5) [Mockbox](#) (5) [SotR](#) (4) [ColdFusion 5](#) (3) [Snake Oil](#) (3) [ColdFusion 8](#) (2) [Frameworks](#) (2) [guest author](#) (2) [A sheep](#) (1)

### PEOPLE

[Sean Corfield](#) (26) [Ray Camden](#) (20) [Adam Tuttle](#)

struct	
I	2
V	b

struct	
I	3
V	c

struct	
I	4
V	d

(One doesn't need to specify both arguments; just the index one is fine. This is a bug, and I will raise it accordingly: [3713035](#). `listEach()` has the same problem: one *needs* to specify the argument in the callback definition, or the function doesn't work).

Anyway, it's a good enhancement to ColdFusion 11 that the callback also receives the index as well the value.

### filter()

The filter methods also iterate over the given object and returns a new object. The callback returns a boolean which determines whether the current element is returned in the new object, eg:

```
numbers = "1,2,3,4";
odds = listFilter(numbers, function(v){
    return v MOD 2;
});
writeDump([ {numbers=numbers}, {odds=odds} ]);
```

Here the callback returns true for each odd list element, so we end up with a list with just the odd numbers in it:

array					
1	<table><tr><th colspan="2">struct</th></tr><tr><td>NUMBERS</td><td>1,2,3,4</td></tr></table>	struct		NUMBERS	1,2,3,4
struct					
NUMBERS	1,2,3,4				
2	<table><tr><th colspan="2">struct</th></tr><tr><td>ODDS</td><td>1,3</td></tr></table>	struct		ODDS	1,3
struct					
ODDS	1,3				

So those are the ones from ColdFusion 10. Old news.

### map()

The `map()` functions iterate over the collection (be it a list, array or struct), and returns a new object with an element for each of the ones in the original collection. The callback in this case returns the *new* element for the new collection, which is derived from the original collection. So it *remaps* the original collection. Here's examples of each of them:

#### listMap()

This function doesn't work, I'm afraid (or I'm doing something wrong which I cannot identify). We're not off to a good start here. Here's some sample code:

```
rainbow = "Whero,Karaka,Kowhai,Kakariki,Kikorangi,Tawatawa,Mawhero";

externalList = "";
reverseRainbow = listMap(rainbow,function(V,I,L){
    var newValue = "#I#:#V.reverse()#";
    externalList = externalList.append(newValue);
    return newValue;
});
writeDump([ {rainbow=rainbow}, {reverseRainbow=reverseRainbow}, {externalList=externalList}
]);

externalList = "";
reverseRainbow = rainbow.map(function(v,i,l){
    var newValue = "#i#:#v.reverse()#";
    externalList = externalList.append(newValue);
    return newValue;
});
writeDump([ {rainbow=rainbow}, {reverseRainbow=reverseRainbow}, {externalList=externalList}
]);
```

(18) [Rakshith Naresh](#) (15) [Brad Wood](#) (12) [Russ Michaels](#) (12) [Scott Stroz](#) (11) [Dave Ferguson](#) (10) [Alex Skinner](#) (9) [Mark Drew](#) (9) [Simon Baynes](#) (9) [Andrew Myers](#) (8) [Kai Koenig](#) (8) [Micha Offner-Streit](#) (8) [Rupesh Kumar](#) (8) [Andrew Scott](#) (7) [Gavin Pickin](#) (7) [Matt Bourke](#) (7) [Bruce Kirkpatrick](#) (6) [Duncan Cumming](#) (6) [Gert Franz](#) (6) [Luis Majano](#) (6) [Matt Gifford](#) (6) [Rob Glover](#) (6) [Carol Hamilton](#) (5) [Frank Jennings](#) (5) [Henry Ho](#) (5) [Mike Hnat](#) (5) [Chris Kobrzak](#) (4) [Dave McGuigan](#) (4) [James Moberg](#) (4) [Jason Dean](#) (4) [Mark Mandel](#) (4) [Aurelien Deleusiére](#) (3) [Brian Sadler](#) (3) [Charlie Arehart](#) (3) [Dale Fraser](#) (3) [Jay Cunningham](#) (3) [Kurt Wiersma](#) (3) [Richard Herbert](#) (3) [Sharon DiOrio](#) (3) [Shawn Holmes](#) (3) [Shilpi Khariwal](#) (3) [Steve Neiland](#) (3)

### BLOG ARCHIVE

#### ▼ 2014 (70)

##### ▼ February (34)

[ColdFusion 11: cfhtmltopdf a non-starter. Literall...](#)

[ColdFusion 11: .map\(\) and .reduce\(\)](#)

[ColdFusion 11: lists and arrays and empty elements...](#)

[ColdFusion 11: query column types preserved when s...](#)

[Five-tagger? what n-tagger am I?](#)

[Breaking out of an each\(\) loop](#)

[ColdFusion 11: a lot of string member functions ha...](#)

[ColdFusion 11: preserveCaseForStructKey](#)

[ColdFusion 11: member functions implementations an...](#)

[Fixing any bug has backwards compatibility concern...](#)

[Feature toggling for both Railo and ColdFusion](#)

[Expressions and operators and doing weird shit](#)

[ColdFusion 11: queryExecute\(\)](#)

[Can we please agree that Adobe is not the arbitor ...](#)

[Completely off topic: this is the state of spam](#)

[ColdFusion 11: good stuff](#)

[ColdFusion 11: preventing files from being include...](#)

[ColdFusion 11: @cfmlnotifier feeds updated](#)

[ColdFusion 11: first bug. Bad bug.](#)

[ColdFusion 11: "Getting Started Server"](#)

[ColdFusion 11 has gone public beta...](#)

[TestBox, BDD-style tests, Railo member functions a...](#)

[ColdFusion-UI-the-Right-Way: <cfajaxproxy>](#)

[I might not be, but Gavin is...](#)

[Slow news day & Adobe charging twice for CFML feat...](#)

[Railo bug? Or ColdFusion bug...](#)

[Things I am not...](#)

This contains the same example using both the `listMap()` function, and the `.map()` method. Here's the output:

array					
1	<table><tr><th colspan="2">struct</th></tr><tr><td>RAINBOW</td><td>Whero, Karaka, Kowhai, Kakariki, Kikorangi, Tawatawa, Mawhero</td></tr></table>	struct		RAINBOW	Whero, Karaka, Kowhai, Kakariki, Kikorangi, Tawatawa, Mawhero
struct					
RAINBOW	Whero, Karaka, Kowhai, Kakariki, Kikorangi, Tawatawa, Mawhero				
2	<table><tr><th colspan="2">struct</th></tr><tr><td>REVERSERAINBOW</td><td>Whero, Karaka, Kowhai, Kakariki, Kikorangi, Tawatawa, Mawhero</td></tr></table>	struct		REVERSERAINBOW	Whero, Karaka, Kowhai, Kakariki, Kikorangi, Tawatawa, Mawhero
struct					
REVERSERAINBOW	Whero, Karaka, Kowhai, Kakariki, Kikorangi, Tawatawa, Mawhero				
3	<table><tr><th colspan="2">struct</th></tr><tr><td>EXTERNALLIST</td><td>1: orehW, 2: akaraK, 3: iahwoK, 4: ikirakaK, 5: ignarokIK, 6: awatawaT, 7: orehwaM</td></tr></table>	struct		EXTERNALLIST	1: orehW, 2: akaraK, 3: iahwoK, 4: ikirakaK, 5: ignarokIK, 6: awatawaT, 7: orehwaM
struct					
EXTERNALLIST	1: orehW, 2: akaraK, 3: iahwoK, 4: ikirakaK, 5: ignarokIK, 6: awatawaT, 7: orehwaM				
array					
1	<table><tr><th colspan="2">struct</th></tr><tr><td>RAINBOW</td><td>Whero, Karaka, Kowhai, Kakariki, Kikorangi, Tawatawa, Mawhero</td></tr></table>	struct		RAINBOW	Whero, Karaka, Kowhai, Kakariki, Kikorangi, Tawatawa, Mawhero
struct					
RAINBOW	Whero, Karaka, Kowhai, Kakariki, Kikorangi, Tawatawa, Mawhero				
2	<table><tr><th colspan="2">struct</th></tr><tr><td>REVERSERAINBOW</td><td>Whero, Karaka, Kowhai, Kakariki, Kikorangi, Tawatawa, Mawhero</td></tr></table>	struct		REVERSERAINBOW	Whero, Karaka, Kowhai, Kakariki, Kikorangi, Tawatawa, Mawhero
struct					
REVERSERAINBOW	Whero, Karaka, Kowhai, Kakariki, Kikorangi, Tawatawa, Mawhero				
3	<table><tr><th colspan="2">struct</th></tr><tr><td>EXTERNALLIST</td><td>1: orehW, 2: akaraK, 3: iahwoK, 4: ikirakaK, 5: ignarokIK, 6: awatawaT, 7: orehwaM</td></tr></table>	struct		EXTERNALLIST	1: orehW, 2: akaraK, 3: iahwoK, 4: ikirakaK, 5: ignarokIK, 6: awatawaT, 7: orehwaM
struct					
EXTERNALLIST	1: orehW, 2: akaraK, 3: iahwoK, 4: ikirakaK, 5: ignarokIK, 6: awatawaT, 7: orehwaM				

I've added in the `externalList` to show you what `reverseRainbow` *should* look like. `listMap()` is supposed to work that the callback receives the element `value`, its `index`, and the `whole list` as arguments. Then it uses that information however is appropriate to create and return a `new element`. Here the new element is very contrived: the `element index`, and its `value` (reversed).

However the original list is just being returned by `listMap()` here. That's wrong. Bug: [3713038](#).

Also note `listMap()` takes some other arguments I'm not using here: the list delimiter and a flag as to whether to respect empty list items (this is like most/all other list functions). But there's another slight glitch here: those values should *also* be passed to the callback, as they might be necessary for doing the element remapping. Bug: [3713043](#).

It looks like we're off to a shocking start here, but that's the end of the bugs I found.

#### `arrayMap()`

In this example we can get a better idea of how the mapping process is supposed to work. Here's an example using both the function and the method:

```
rainbow = ["Whero", "Karaka", "Kowhai", "Kakariki", "Kikorangi", "Tawatawa", "Mawhero"];
colourInList = arrayMap(
    rainbow,
    function(v,i,a){
        return replace(a.toList(), v, ucase(v));
    }
);
writeDump([rainbow, colourInList]);

rainbow.map(function(v,i,a){
    return replace(a.toList(), v, ucase(v));
});
writeDump([rainbow, colourInList]);
```

array																
1	<table><tr><th>array</th></tr><tr><td>1</td><td>Whero</td></tr><tr><td>2</td><td>Karaka</td></tr><tr><td>3</td><td>Kowhai</td></tr><tr><td>4</td><td>Kakariki</td></tr><tr><td>5</td><td>Kikorangi</td></tr><tr><td>6</td><td>Tawatawa</td></tr><tr><td>7</td><td>Mawhero</td></tr></table>	array	1	Whero	2	Karaka	3	Kowhai	4	Kakariki	5	Kikorangi	6	Tawatawa	7	Mawhero
array																
1	Whero															
2	Karaka															
3	Kowhai															
4	Kakariki															
5	Kikorangi															
6	Tawatawa															
7	Mawhero															

[Installing and Configuring Apache 2.2, Tomcat 6.0,...](#)

[ColdFusion 9 on Windows 8](#)

[ColdFusion-UI-the-Right-Way: <cfchart>](#)

[Waitangi Day again](#)

[Cheers lads](#)

[Now, children...](#)

[All shouty and bitey... and it's not even me doing...](#)

► [January](#) (36)

► [2013](#) (384)

► [2012](#) (143)

**FOLLOW BY EMAIL**

**FOLLOWERS**

**CRAP I SPOUT ON TWITTER**

[Tweets by @daccf](#)

2

array

1

Whero, Karaka, Kowhai, Kakariki, Kikorangi, Tawatawa, Mawhero

2

Whero, KARAKA, Kowhai, Kakariki, Kikorangi, Tawatawa, Mawhero

3

Whero, Karaka, KOWHAI, Kakariki, Kikorangi, Tawatawa, Mawhero

4

Whero, Karaka, Kowhai, KAKARIKI, Kikorangi, Tawatawa, Mawhero

5

Whero, Karaka, Kowhai, Kakariki, KIKORANGI, Tawatawa, Mawhero

6

Whero, Karaka, Kowhai, Kakariki, Kikorangi, TAWATAWA, Mawhero

7

Whero, Karaka, Kowhai, Kakariki, Kikorangi, Tawatawa, MAWHERO

array

1

array

1

Whero

2

Karaka

3

Kowhai

4

Kakariki

5

Kikorangi

6

Tawatawa

7

Mawhero

2

array

1

Whero, Karaka, Kowhai, Kakariki, Kikorangi, Tawatawa, Mawhero

2

Whero, KARAKA, Kowhai, Kakariki, Kikorangi, Tawatawa, Mawhero

3

Whero, Karaka, KOWHAI, Kakariki, Kikorangi, Tawatawa, Mawhero

4

Whero, Karaka, Kowhai, KAKARIKI, Kikorangi, Tawatawa, Mawhero

5

Whero, Karaka, Kowhai, Kakariki, KIKORANGI, Tawatawa, Mawhero

6

Whero, Karaka, Kowhai, Kakariki, Kikorangi, TAWATAWA, Mawhero

7

Whero, Karaka, Kowhai, Kakariki, Kikorangi, Tawatawa, MAWHERO

Here the callback receives the array element **value**, its **index**, and the **entire array**. From this, we convert the **array to a list**, and then **highlight (with capitals)** the **current element** in that list, returning the whole list. So the resultant remapped array contains an element for each original element, but completely different data than the original array; with each new element being that remapping done in the callback. And the method version works exactly the same (I'm both demonstrating and testing here too, hence the double-up).

If you have a sparse array - one without an element at each index - you need to deal with this by hand. The iteration is index-centric, not element-centric, so each index will have the callback called on it, so you need to deal with the possibility of no *value* being passed to the callback:

```
a = [1];
a[3] = 3;
writeDump(a);

result = a.map(function(v,i,a){
    if (structKeyExists(arguments, "v")){
        return v^2;
    }
});
writeDump(result);

result = a.map(function(v=0,i,a){
    return v^2;
});
writeDump(result);
```

Output:

array	
1	1
2	[undefined array element] Element 2 is undefined in a Java object of type class coldfusion.runtime.Array.
3	3

array	
1	1
2	[undefined array element] Element 2 is undefined in a Java object of type class coldfusion.runtime.Array.
3	9

array	
1	1

2	0
3	9

Here I am using two different techniques. In the first version I am simply **checking to see if the value exists**, and only returning a mapped value if so. In the second version I am **defaulting the value in the callback definition**. It's really situation-dependent as to which approach to take. In this case, the second approach is *not* really appropriate.

#### structMap()

This is more of the same really. Here the callback receives the key and the value:

```
original = {"one"={1="tahi"},"two"={2="rua"},"three"={3="toru"},"four"={4="wha"}};
fixed = structMap(original, function(k,v){
    return v[v.keyList().first()];
});
writeDump([original,fixed]);

fixed = original.map(function(k,v){
    return v.keyList().first();
});
writeDump([original,fixed]);
```

array																											
1	<table> <tr><th colspan="2">struct</th></tr> <tr> <td>four</td><td> <table> <tr><th colspan="2">struct</th></tr> <tr><td>4</td><td>wha</td></tr> </table> </td></tr> <tr> <td>one</td><td> <table> <tr><th colspan="2">struct</th></tr> <tr><td>1</td><td>tahi</td></tr> </table> </td></tr> <tr> <td>three</td><td> <table> <tr><th colspan="2">struct</th></tr> <tr><td>3</td><td>toru</td></tr> </table> </td></tr> <tr> <td>two</td><td> <table> <tr><th colspan="2">struct</th></tr> <tr><td>2</td><td>rua</td></tr> </table> </td></tr> </table>	struct		four	<table> <tr><th colspan="2">struct</th></tr> <tr><td>4</td><td>wha</td></tr> </table>	struct		4	wha	one	<table> <tr><th colspan="2">struct</th></tr> <tr><td>1</td><td>tahi</td></tr> </table>	struct		1	tahi	three	<table> <tr><th colspan="2">struct</th></tr> <tr><td>3</td><td>toru</td></tr> </table>	struct		3	toru	two	<table> <tr><th colspan="2">struct</th></tr> <tr><td>2</td><td>rua</td></tr> </table>	struct		2	rua
struct																											
four	<table> <tr><th colspan="2">struct</th></tr> <tr><td>4</td><td>wha</td></tr> </table>	struct		4	wha																						
struct																											
4	wha																										
one	<table> <tr><th colspan="2">struct</th></tr> <tr><td>1</td><td>tahi</td></tr> </table>	struct		1	tahi																						
struct																											
1	tahi																										
three	<table> <tr><th colspan="2">struct</th></tr> <tr><td>3</td><td>toru</td></tr> </table>	struct		3	toru																						
struct																											
3	toru																										
two	<table> <tr><th colspan="2">struct</th></tr> <tr><td>2</td><td>rua</td></tr> </table>	struct		2	rua																						
struct																											
2	rua																										
2	<table> <tr><th colspan="2">struct</th></tr> <tr><td>four</td><td>wha</td></tr> <tr><td>one</td><td>tahi</td></tr> <tr><td>three</td><td>toru</td></tr> <tr><td>two</td><td>rua</td></tr> </table>	struct		four	wha	one	tahi	three	toru	two	rua																
struct																											
four	wha																										
one	tahi																										
three	toru																										
two	rua																										

array																											
1	<table> <tr><th colspan="2">struct</th></tr> <tr> <td>four</td><td> <table> <tr><th colspan="2">struct</th></tr> <tr><td>4</td><td>wha</td></tr> </table> </td></tr> <tr> <td>one</td><td> <table> <tr><th colspan="2">struct</th></tr> <tr><td>1</td><td>tahi</td></tr> </table> </td></tr> <tr> <td>three</td><td> <table> <tr><th colspan="2">struct</th></tr> <tr><td>3</td><td>toru</td></tr> </table> </td></tr> <tr> <td>two</td><td> <table> <tr><th colspan="2">struct</th></tr> <tr><td>2</td><td>rua</td></tr> </table> </td></tr> </table>	struct		four	<table> <tr><th colspan="2">struct</th></tr> <tr><td>4</td><td>wha</td></tr> </table>	struct		4	wha	one	<table> <tr><th colspan="2">struct</th></tr> <tr><td>1</td><td>tahi</td></tr> </table>	struct		1	tahi	three	<table> <tr><th colspan="2">struct</th></tr> <tr><td>3</td><td>toru</td></tr> </table>	struct		3	toru	two	<table> <tr><th colspan="2">struct</th></tr> <tr><td>2</td><td>rua</td></tr> </table>	struct		2	rua
struct																											
four	<table> <tr><th colspan="2">struct</th></tr> <tr><td>4</td><td>wha</td></tr> </table>	struct		4	wha																						
struct																											
4	wha																										
one	<table> <tr><th colspan="2">struct</th></tr> <tr><td>1</td><td>tahi</td></tr> </table>	struct		1	tahi																						
struct																											
1	tahi																										
three	<table> <tr><th colspan="2">struct</th></tr> <tr><td>3</td><td>toru</td></tr> </table>	struct		3	toru																						
struct																											
3	toru																										
two	<table> <tr><th colspan="2">struct</th></tr> <tr><td>2</td><td>rua</td></tr> </table>	struct		2	rua																						
struct																											
2	rua																										
2	<table> <tr><th colspan="2">struct</th></tr> <tr><td>four</td><td>4</td></tr> <tr><td>one</td><td>1</td></tr> <tr><td>three</td><td>3</td></tr> <tr><td>two</td><td>2</td></tr> </table>	struct		four	4	one	1	three	3	two	2																
struct																											
four	4																										
one	1																										
three	3																										
two	2																										

Here the value being passed into the callback is the substruct with the digit as a key and the Maori number for a value. I am using that information to map to a struct which has just the Maori version as the new struct's value (the `structMap()` example); and in the `.map()` example I'm just mapping to a struct with the digit as values. These are pretty contrived examples, but you hopefully get the idea.

## `.reduce()`

The `reduce()` operation is slightly more complex. Basically it iterates over the collection and from each element of the collection, derives one single value as a result.

## `listReduce()`

Here we perform a sum and a product on a list of digits:

```
numbers = "1,2,3,4,5,6,7,8,9,10";

sum = listReduce(
  numbers,
  function(previousValue, value){
    return previousValue + value;
  },
  0
);
writeOutput("The sum of the digits #numbers# is #sum#<br>");

product = numbers.reduce(
  function(previousValue, value){
    return previousValue * value;
  },
  1
);
writeOutput("The product of the digits #numbers# is #product#<br>");
```

Note that the callback receives two arguments: the `previous value`, and the `current value`. And also note the function can accept a `starting value` too. That said, it doesn't *need* to take a starting value, but if you don't give it one, you have to deal with not receiving a value for it in the first iteration. One can handle this like this:

```
numbers = "1,2,3,4,5,6,7,8,9,10";

sum = numbers.reduce(function(previousValue, value){
  if (!structKeyExists(arguments, "previousValue")){
    return value;
  }
  return previousValue + value;
});
writeOutput("The sum of the digits #numbers# is #sum#<br>");
```

Or like this:

```
product = numbers.reduce(function(previousValue=1, value){
  return previousValue * value;
});
writeOutput("The product of the digits #numbers# is #product#<br>");
```

To be honest, given one can default the `previousValue` argument in the callback definition, I wonder if the `listReduce()` function itself *needs* to have that `initialValue` argument? It seems like pointless duplication to me, perhaps? Hopefully someone who has more experience with these functions (Adam Tuttle and Sean, I am looking at you two), and can advise where one or other approach might be better.

Oh... and the output from all this (for the sake of completeness):

The sum of the digits 1,2,3,4,5,6,7,8,9,10 is 55  
The product of the digits 1,2,3,4,5,6,7,8,9,10 is 3628800

## `arrayReduce()`

This works the same way:

```
rainbow = ["Whero", "Karaka", "Kowhai", "Kakariki", "Kikorangi", "Tawatawa", "Mawhero"];

ul = arrayReduce(
  rainbow,
  function(previousValue, value){
    return previousValue & "<li>#value#</li>";
  },
  "<ul>"
);
```

```

) & "</ul>";
writeOutput(ul);

ol = rainbow.reduce(
  function(previousValue, value){
    return previousValue & "<li>#value#</li>";
  },
  "<ol>"
) & "</ol>";
writeOutput(ol);

```

Actually this is probably a good demonstration of the subtle difference between using a starting value and defaulting the previous value. `<ul>` makes a sensible *starting* value, perhaps; but does not make sense as a default *previous* value.

And the output this time is just some mark-up:

- Whero
  - Karaka
  - Kowhai
  - Kakariki
  - Kikorangi
  - Tawatawa
  - Mawhero
1. Whero
  2. Karaka
  3. Kowhai
  4. Kakariki
  5. Kikorangi
  6. Tawatawa
  7. Mawhero

(and, yes, I know I could have just generated the `<li>` tags with the reduction, then slapped the `<ul>` and `<ol>` around them afterwards, but that's not the point ;-)

#### structReduce()

This example is very similar to the previous one:

```

rainbow = {
  "Red"="Whero",
  "Orange"="Karaka",
  "Yellow"="Kowhai",
  "Green"="Kakariki",
  "Blue"="Kikorangi",
  "Indigo"="Tawatawa",
  "Pink"="Mawhero"
};

dl = structReduce(
  rainbow,
  function(previousValue, key, value){
    return previousValue & "<dt>#key#</dt><dd>#value#</dd>";
  },
  "<dl>"
) & "</dl>";
writeOutput(dl);

dl = rainbow.reduce(
  function(previousValue, key, value){
    return previousValue & "<dt>#value#</dt><dd>#key#</dd>";
  },
  "<dl>"
) & "</dl>";
writeOutput(dl);

```

Output:

```

Blue    Kikorangi
Yellow  Kowhai
Green   Kakariki
Pink    Mawhero
Indigo  Tawatawa
Orange  Karaka

```

**Red**      Whero  
  
**Kikorangi** Blue  
**i**      **Kowhai**   Yellow  
**Kakariki** Green  
**Mawhera** Pink  
**o**      **Tawata**   Indigo  
         **wa**      **Karaka**   Orange  
**Whero**   Red

Here I just demonstrate how I'm using both the `key` and `value` from the struct.

That's about it.

Do you know what I am left wondering though... where are the equivalent methods for query objects? They're collections too, after all. I better get a ticket in to get those provided for too: [3713323](#).

Bed time for me. It's been a very bloody long ColdFusion 11 day for me. I'm scooting back to the UK tomorrow... 30-odd hours worth of aircraft and airports. Joy. At least once I get back I then have 1.5h on the train too. This will quite possibly mark the end of the torrent of ColdFusion 11 stuff from me. I've done my best to blog as much as I can whilst I've been off work, but after today I'll be indisposed, in Ireland, or back at work next Monday. So I'll probably drop back to around one article per day, tops.

--

Adam

Posted by [Adam Cameron](#) at 09:20



[8 Comments](#)

Labels: [ColdFusion 11](#)

[Newer Post](#)

[Home](#)

[Older Post](#)

Subscribe to: [Post Comments \(Atom\)](#)

#### SUBSCRIBE TO

 [Posts](#) 

 [Comments](#) 