LEARNING coldfusion

#coldfusion

Table of Contents

About

Chapter 1: Getting started with coldfusion

Remarks

This section provides an overview of what coldfusion is, and why a developer might want to use it.

It should also mention any large subjects within coldfusion, and link out to the related topics. Since the Documentation for coldfusion is new, you may need to create initial versions of those related topics.

Versions

Version	Release Date
Cold Fusion version 1.0	1995-07-02
Cold Fusion version 1.5	1996-01-01
Cold Fusion version 2.0	1996-10-01
Cold Fusion version 3.0	1997-06-01
Cold Fusion version 3.1	1998-01-01
ColdFusion version 4.0	1998-11-01
ColdFusion version 4.5.1	1999-11-01
ColdFusion version 5.0	2001-06-01
ColdFusion MX version 6.0	2002-05-01
ColdFusion MX version 6.1	2003-07-01
ColdFusion MX 7	2005-02-07
ColdFusion 8	2007-07-30
ColdFusion 9	2009-10-05
ColdFusion 10	2012-05-15
ColdFusion 11	2014-04-29
ColdFusion 2016	2016-02-16

Examples

Installation or Setup

Linux (Ubuntu) Installation

Lucee (Open Source)

ColdFusion / CFML Interpretor

Download the appropriate file from their site (http://lucee.org/downloads.html) and execute their installer

```
wget http://cdn.lucee.org/downloader.cfm/id/155/file/lucee-5.0.0.252-pl0-linux-x64-
installer.run
sudo chmod +x lucee-5.0.0.252-pl0-linux-x64-installer.run
sudo ./lucee-5.0.0.252-pl0-linux-x64-installer.run
```

Step through installer.

Nginx

Install Nginx on your server

```
sudo apt-get install nginx
```

Edit your /etc/nginx/sites-available/default

```
server {
    listen 80;
    server_name _;

    root /opt/lucee/tomcat/webapps/ROOT;
    index index.cfm index.html index.htm;

    #Lucee Admin should always proxy to Lucee
    location /lucee {
        include lucee.conf;
    }

    #Pretty URLs
    location / {
        try_files $uri /index.cfm$uri?$is_args$args;
        include lucee.conf;
    }

    location ~ \.cfm {
        include lucee.conf;
    }
    location ~ \.cfc {
        include lucee.conf;
    }
}
```

Edit /etc/nginx/lucee.conf

```
proxy_pass http://127.0.0.1:8888;
proxy_set_header Host $http_host;
proxy_set_header X-Real-IP $remote_addr;
proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
proxy_set_header X-Forwarded-Proto $scheme;
```

Reload nginx

```
sudo service nginx reload
```

Access the Lucee Server admin here:

```
127.0.0.1/lucee/admin/server.cfm
```

or

```
127.0.0.1:8888/lucee/admin/server.cfm
```

Your root web directory lives here:

```
/opt/lucee/tomcat/webapps/ROOT
```

Adobe (Closed Source)

ColdFusion / CFML Interpretor

Download the appropriate file from their site (https://www.adobe.com/products/coldfusion/download-trial/try.html) and execute their installer

```
wget <URL>/ColdFusion_2016_WWEJ_linux64.bin
sudo chmod +x ColdFusion_2016_WWEJ_linux64.bin
sudo ./ColdFusion_2016_WWEJ_linux64.bin
```

Step through installer. Make sure you select the internal web server (port 8500)

Nginx

Install Nginx on your server

```
sudo apt-get install nginx
```

Edit your /etc/nginx/sites-available/default

```
server {
    listen 80;
    server_name _;

    root /opt/coldfusion2016/cfusion/wwwroot;
    index index.cfm index.html index.htm;

    location / {
        try_files $uri $uri/ =404;
    }

    location ^~ /CFIDE/administrator {
        deny all;
    }

    location ~* \.(cfm|cfml|cfc|html)$ {
        include /etc/nginx/conf/dc_tomcat_connector.conf;
    }

    location ^~ /rest {
        include tomcatconf;
    }
}
```

Edit /etc/nginx/tomcat.conf

```
proxy_pass http://127.0.0.1:8500;
proxy_set_header Host $host;
proxy_set_header X-Forwarded-Host $host;
proxy_set_header X-Forwarded-Server $host;
proxy_set_header X-Forwarded-For $http_x_forwarded_for;
proxy_set_header X-Real-IP $remote_addr;
```

Reload nginx

```
sudo service nginx reload
```

Access the Adobe ColdFusion Server admin here:

```
127.0.0.1:8500/CFIDE/administrator/index.cfm
```

Your root web directory lives here:

```
/opt/coldfusion2016/cfusion/wwwroot
```

Hello World

File: test.cfm

Tag Implementation

```
<cfoutput>Hello World!</cfoutput>
```

CFScript Implementation

```
<cfscript>
writeOutput("Hello World!");
</cfscript>
```

Read Getting started with coldfusion online: https://riptutorial.com/coldfusion/topic/913/getting-started-with-coldfusion

Chapter 2: CFLOOP How-To

Remarks

Big thanks to

- Pete Freitag for his CFScript Cheat Sheet
- Adam Cameron for CF 11: CFLOOP in CFScript is Very Broken (and it still is in CF 2016).

Examples

Looping through a collection using CFML tags.

```
<!--- Define collection --->
<cfset attributes = {
  "name": "Sales",
  "type": "text",
  "value": "Connection"
}>

<!---
  cfloop tag with attribute collection can be used to
  loop through the elements of a structure
--->
<cfloop collection=#attributes# item="attribute">
  <cfoutput>
    Key : #attribute#, Value : #attributes[attribute]#
  </cfoutput>
</cfloop>
```

Looping through a collection using CFSCRIPT.

```
<cfscript>
  /*define collection*/
  attributes = {
    "name": "Sales",
    "type": "text",
    "value": "Connection"
  };
  for(attribute in attributes){
    /* attribute variable will contain the key name of each key value pair in loop */
    WriteOutput('Key : ' & attribute & ', Value : ' & attributes[attribute] & '<br/>');
  }
</cfscript>
```

Index

Parameters

Attribute	Required	Type	Default	Description
index	true	string		Variable name for the loop's index. Defaults to the `variables` scope.
from	true	numeric		Starting value for the index.
to	true	numeric		Ending value for the index.
step	false	numeric	1	Value by which to increase or decrease the index per iteration.

Basic index loop

Final value of `x` is 10.

```
<!--- Tags --->
<cfoutput>
    <cfloop index="x" from="1" to="10">
        <li>#x#</li>
    </cfloop>
</cfoutput>
<!--- cfscript --->
<cfscript>
    for (x = 1; x <= 10; x++) {
        writeOutput('<li>' & x & '</li>');
    }
</cfscript>
<!--- HTML Output --->
 - 1
 - 2
 - 3
 - 4
 - 5
 - 6
 - 7
 - 8
 - 9
 - 10
```

Increase step to 2

Final value of `x` is 11.

```
<!--- Tags --->
<cfoutput>
    <cfloop index="x" from="1" to="10" step="2">
        <li>#x#</li>
    </cfloop>
</cfoutput>
<!--- cfscript --->
<cfscript>
    for (x = 1; x <= 10; x += 2) {
        writeOutput('<li>' & x & '</li>');
    }
</cfscript>
<!--- HTML Output --->
 - 1
 - 3
 - 5
 - 7
 - 9
```

Decrement step by 1

Final value of `x` is 0.

```
<!--- Tags --->
<cfoutput>
    <cfloop index="x" from="10" to="1" step="-1">
        <li>#x#</li>
    </cfloop>
</cfoutput>
<!--- cfscript --->
<cfscript>
    for (x = 10; x > 0; x--) {
        writeOutput('<li>' & x & '</li>');
    }
</cfscript>
<!--- HTML Output --->
 - 10
 - 9
 - 8
 - 7
 - 6
 - 5
 - 4
 - 3
 - 2
 - 1
```

CFLoop in a Function

Make sure to `var` or `local` scope the index inside a function. `Foo()` returns 11.

```
<!--- var scope --->
<cffunction name="foo" access="public" output="false" returntype="numeric">
    <cfset var x = 0 />
    <cfloop index="x" from="1" to="10" step="1">
        <cfset x++ />
    </cfloop>
    <cfreturn x />
</cffunction>

<!--- Local scope --->
<cffunction name="foo" access="public" output="false" returntype="numeric">
    <cfloop index="local.x" from="1" to="10" step="1">
        <cfset local.x++ />
    </cfloop>
    <cfreturn local.x />
</cffunction>
```

ColdFusion 11 through current

> The cfscript function `cfloop` has no support for `index` as a stand alone counter mechanism.

Condition

Tag syntax

Parameters

Attribute	Required	Type	Default	Description
condition	true	string		Condition that manages the loop. Cannot contain math symbols like `<`, `>` or `=`. Must use ColdFusion text implementations like `less than`, `lt`, `greater than`, `gt`, `equals` or `eq`.

Final value of `x` is 5.

```
<cfset x = 0 />
<cfoutput>
    <cfloop condition="x LT 5">
        <cfset x++ />
        <li>#x#</li>
    </cfloop>
</cfoutput>
```

Generated HTML

This will also have a line break between each line of HTML.

```
<li>1</li>
<li>2</li>
<li>3</li>
<li>4</li>
<li>5</li>
```

CFScript

Previous to ColdFusion 8

```
<cfscript>
x = 0;
while (x LT 5) {
    x = x + 1;
    writeOutput('<li>' & x & '</li>');
}
</cfscript>
```

ColdFusion 8 through current

```
<cfscript>
x = 0;
while (x LT 5) {
    x = x++;
    writeOutput('<li>' & x & '</li>');
}
</cfscript>
```

ColdFusion 11 through current

> The cfscript function `cfloop` has no support for `condition`.

Generated HTML

Notice that the cfscript output is all on one line.

```
<li>one</li><li>two</li><li>three</li><li>four</li>
```

Date or time range

Example for date or time range.

Query

Consider the table `dbo.state_zip`, which contains the columns `city`, `statecode` and `zipcode` and has over 80,000 records.

Parameters

Attribute	Required	Type	Default	Description
query	true	string		The variable name of a query object.
startrow	false	numeric		The starting row index of the query object.

Attribute	Required	Type	Default	Description
endrow	false	numeric		The ending row index of the query object.
group	false	string		The query column name on which to group records.

Example query

```
<cfquery name="geo" datasource="reotrans-dev">
    SELECT city, stateCode, zipCode
    FROM dbo.state_zip
</cfquery>
```

Tag syntax

Using the query object `geo` as the source for `cfloop`. Since the table `dbo.state_zip` has so many records, the HTML generated will take quite some time. This example shows only the first 20 records' worth of HTML.

```
<cfoutput>
    <ul>
    <cfloop query="geo">
        <!--- Scope the column names with the query name. --->
        <li>#geo.city# | #geo.stateCode# | #geo.zipCode#</li>
    </cfloop>
    </ul>
</cfoutput>
```

Generated HTML

```
<ul>
    <li>100 PALMS | CA | 92274</li>
    <li>1000 PALMS | CA | 92276</li>
    <li>12 MILE | IN | 46988</li>
    <li>1ST NATIONAL BANK OF OMAHA | NE | 68197</li>
    <li>29 PALMS | CA | 92277</li>
    <li>29 PALMS | CA | 92278</li>
    <li>3 STATE FARM PLAZA | IL | 61710</li>
    <li>3 STATE FARM PLAZA | IL | 61791</li>
    <li>30TH STREET | PA | 19104</li>
    <li>3M CORP | MN | 55144</li>
    <li>65TH INFANTRY | PR | 00923</li>
    <li>65TH INFANTRY | PR | 00924</li>
    <li>65TH INFANTRY | PR | 00929</li>
    <li>65TH INFANTRY | PR | 00936</li>
    <li>7 CORNERS | VA | 22044</li>
    <li>88 | KY | 42130</li>
    <li>9 MILE POINT | LA | 70094</li>
    <li>A A R P INS | PA | 19187</li>
    <li>A A R P PHARMACY | CT | 06167</li>
    <li>A H MCCOY FEDERAL BLDG | MS | 39269</li>
</ul>
```

Limiting output to specific rows

To limit the query's output to a specific range of rows, specify `startrow` and `endrow`.

```
<cfloop query="geo" startrow="100" endrow="150">
    <li>#geo.city# | #geo.stateCode# | #geo.zipCode#</li>
</cfloop>
```

Grouping Output

In the example data, the same state listed multiple times in relation to the multiple cities that are associated to each state. You can also see the same city listed multiple times in relation to the multiple zip codes associated to each city.

Let's group the output by state first. Notice the 2nd instance of `cfloop` wrapped around the content that will be output under the `stateCode` grouped content.

```
<cfoutput>
    <ul>
    <cfloop query="geo" group="stateCode">
        <!--- Scope the column names with the query name. --->
        <li>#geo.stateCode#
            <ul>
                <cfloop>
                    <li>#geo.city# | #geo.zipCode#</li>
                </cfloop>
            </ul>
        </li>
    </cfloop>
    </ul>
</cfoutput>
```

Generated HTML (extract) from one grouped `cfloop` tag.

```
<ul>
    <li>AK
        <ul>
            <li>KONGIGANAK | 99545</li>
            <li>ADAK | 99546</li>
            <li>ATKA | 99547</li>
            <!-- etc. -->
        </ul>
    </li>
    <li>AL
        <ul>
            <li>ALEX CITY | 35010</li>
            <li>ALEXANDER CITY | 35010</li>
            <li>ALEX CITY | 35011</li>
            <!-- etc. -->
        </ul>
    </li>
    <!-- etc. -->
</ul>
```

Finally, let's group the output by `stateCode`, then by `city` in order to see all the `zipCode` entries per city. Notice the 2nd `cfloop` is now grouped by `city` and a 3rd `cfloop` exists to output the `zipCode` data.

```
<cfoutput>
    <ul>
    <cfloop query="geo" group="stateCode">
        <li>#geo.stateCode#
            <ul>
            <cfloop group="city">
                <li>#geo.city#
                    <ul>
                        <cfloop>
                            <li>#geo.zipCode#</li>
                        </cfloop>
                    </ul>
                </li>
            </cfloop>
            </ul>
        </li>
    </cfloop>
    </ul>
</cfoutput>
```

Generated HTML (extract) from two grouped `cfloop` tags.

```
<ul>
    <li>AK
        <ul>
            <li>ADAK
                <ul>
                    <li>99546</li>
                    <li>99571</li>
                </ul>
            </li>
            <li>AKHIOK
                <ul>
                    <li>99615</li>
                </ul>
            </li>
            <!--- etc. --->
            <li>BARROW
                <ul>
                    <li>99723</li>
                    <li>99759</li>
                    <li>99789</li>
                    <li>99791</li>
                </ul>
            </li>
            <!--- etc. --->
        </ul>
    </li>
    <!--- stateCodes etc. --->
</ul>
```

CFScript

ColdFusion 6 (MX) though current

```
<cfscript>
    for (x = 1; x LTE geo.recordcount; x = x + 1) {
        writeOutput( '<li>' & geo.city[x] & ' | ' &
            geo.stateCode[x] & ' | ' & geo.zipCode[x] & '</li>');
    }
</cfscript>
```

ColdFusion 8 though current

```
<cfscript>
    for (x = 1; x <= geo.recordcount; x++) {
        writeOutput( '<li>' & geo.city[x] & ' | ' &
            geo.stateCode[x] & ' | ' & geo.zipCode[x] & '</li>');
    }
</cfscript>
```

ColdFusion 10 though current

With the `FOR IN` syntax, `x` is a query row object, not the row index.

```
<cfscript>
    for (x in geo) {
        writeOutput( '<li>' & x.city & ' | ' &
            x.stateCode & ' | ' & x.zipCode & '</li>');
    }
</cfscript>
```

ColdFusion 11 though current

ColdFusion 11 allows most tags to be written as cfscript.

```
<cfscript>
    cfloop(query: geo, startrow: 1, endrow: 2) {
        writeOutput( '<li>' & geo.city & ' | ' &
            geo.stateCode & ' | ' & geo.zipCode & '</li>');
    }
</cfscript>
```

With `group`.

```
<cfscript>
    cfloop(query: geo, group: 'city') {
        writeOutput( '<li>' & geo.city & '<ul>');
        cfloop() { // no arguments, just as in the tag syntax.
```

```
            writeOutput('<li>'  & geo.zipCode & '</li>');
        }
        writeOutput('</ul></li>');
    }
</cfscript>
```

List

Consider this list:

```
<cfset foo = "one,two,three,four" />
```

Tag syntax

Parameters

Attribute	Required	Default	Description
list	true		A list object. The variable must be evaluated (wrapped with ##)
index	true		The current element of the list.

```
<cfoutput>
    <cfloop list="#foo#" index="x">
        <li>#x#</li>
    </cfloop>
</cfoutput>
```

Generated HTML

This will also have a line break between each line of HTML.

```
<li>one</li>
<li>two</li>
<li>three</li>
<li>four</li>
```

CFScript

Previous to ColdFusion 8

```
<cfscript>
    for (x = 1; x LTE listLen(foo); x = x + 1) {
        writeOutput("<li>" & listGetAt(foo, x) & "</li>");
    }
</cfscript>
```

ColdFusion 8 through current

```
<cfscript>
    for (x = 1; x <= listLen(foo); x++) {
        writeOutput("<li>" & listGetAt(foo, x) & "</li>");
    }
</cfscript>
```

ColdFusion 9 through current

```
<cfscript>
    for (x in foo) {
        writeOutput("<li>" & x & "</li>");
    }
</cfscript>
```

ColdFusion 11 through current

> The cfscript function `cfloop` has no support for `list`.

Generated HTML

Notice that the cfscript output is all on one line.

```
<li>one</li><li>two</li><li>three</li><li>four</li>
```

Array

> The ability to directly use an `array` object with `cfloop` was added in ColdFusion 8.

Consider this array;

```
<cfset aFoo = [
    "one"
    , "two"
    , "three"
    , "four"
] />
```

Tag syntax

ColdFusion 8 through current

Using the attribute `index` by itself.

Parameters

Attribute	Required	Default	Description
array	true		An array object. The variable must be evaluated (wrapped with ##)
index	true		The current element of the array.

```
<cfoutput>
    <cfloop array="#aFoo#" index="x">
        <li>#x#</li>
    </cfloop>
</cfoutput>
```

Generated HTML

This will also have a line break between each line of HTML.

```
<li>one</li>
<li>two</li>
<li>three</li>
<li>four</li>
```

ColdFusion 2016 through current

> The attribute `item` changes the behavior of `cfloop` as of Coldfusion 2016.

Using the attribute `item` instead of or in addition to `index`.

Parameters

Attribute	Required	Default	Description
array	true		An array object. The variable must be evaluated (wrapped with ##)
item	true		The current element of the array.
index	false		The current index of the array.

```
<cfoutput>
    <cfloop array="#aFoo#" item="x" index="y">
        <li>#x# | #y#</li>
    </cfloop>
</cfoutput>
```

Generated HTML

This will also have a line break between each line of HTML.

```
<li>one | 1</li>
<li>two | 2</li>
<li>three | 3</li>
<li>four | 4</li>
```

CFScript

Previous to ColdFusion 8

```
<cfscript>
for (i = 1; x LTE arrayLen(aFoo); i = i + 1) {
    writeOutput("<li>" & aFoo[i] & "</li>");
}
</cfscript>
```

ColdFusion 8 through current

```
<cfscript>
for (i = 1; i <= arrayLen(aFoo); i = i++) {
    writeOutput("<li>" & aFoo[i] & "</li>");
}
</cfscript>
```

ColdFusion 9 through current

> With the `FOR IN` syntax, x is the current array element, not the array index.

```
<cfscript>
for (x in aFoo) {
    writeOutput("<li>" & x & "</li>");
}
</cfscript>
```

ColdFusion 11 through current

> The cfscript function `cfloop` has no support for `array`.

Generated HTML

Notice that the `cfscript` output is all on one line.

```
<li>one</li><li>two</li><li>three</li><li>four</li>
```

File

```
<cfloop list="#myFile#" index="FileItem" delimiters="#chr(10)##chr(13)#">
  <cfoutput>
   #FileItem#<br />
 </cfoutput>
</cfloop>
```

Structure

Consider this structure:

```
<cfset stFoo = {
    a = "one"
    , b = "two"
    , c = "three"
    , d = "foue"
} />
```

Tag syntax

Parameters

Notice the use of the attribute `item` instead of `index`.

Attribute	Required	Type	Default	Description
collection	true	structure		A `struct` object. The variable must be evaluated (wrapped with ##).
item	true	string		The current structure `key`,

Using Structure Functions

```
<cfoutput>
    <cfloop collection="#stFoo#" item="x">
        <li>#structFind(stFoo, x)#</li>
    </cfloop>
</cfoutput>
```

Implicit Structure Syntax

```
<cfoutput>
    <cfloop collection="#stFoo#" item="x">
```

```
        <li>#stFoo[x]#</li>
    </cfloop>
</cfoutput>
```

Generated HTML

This will also have a line break between each line of HTML.

```
<li>one</li>
<li>two</li>
<li>three</li>
<li>four</li>
```

CFScript

With the `FOR IN` syntax, `x` is a `key` of the structure object.

Output the structure's keys

```
<cfscript>
    for (x in stFoo) {
        writeOutput("<li>" & x & "</li>");
    }
</cfscript>
```

Generated HTML

```
<li>A</li><li>B</li><li>C</li><li>D</li>
```

Output the value of the structure's keys

Using Structure Functions

```
<cfscript>
    for (x in stFoo) {
        writeOutput("<li>" & structFind(stFoo, x) & "</li>");
    }
</cfscript>
```

Implicit Structure Syntax

```
<cfscript>
    for (x in stFoo) {
        writeOutput("<li>" & stFoo[x] & "</li>");
    }
</cfscript>
```

ColdFusion 11 through current

> The cfscript function `cfloop` has no support for `collection`.

Generated HTML

Notice that the cfscript output is all on one line.

```
<li>one</li><li>two</li><li>three</li><li>four</li>
```

Index Loop

Use the `from` and `to` attributes to specify how many iterations should occur. The (optional) `step` attribute allows you to determine how big the increments will be.

```
<cfloop from="1" to="10" index="i" step="2">
    <cfoutput>
        #i#<br />
    </cfoutput>
</cfloop>
```

Conditional Loop

You use the `condition` attribute to specify the condition to use.

```
<cfset myVar=false>
<cfloop condition="myVar eq false">
    <cfoutput>
    myVar = <b>#myVar#</b>  (still in loop)<br />
 </cfoutput>
    <cfif RandRange(1,10) eq 10>
        <cfset myVar="true">
    </cfif>
</cfloop>
<cfoutput>
    myVar = <b>#myVar#</b> (loop has finished)
</cfoutput>
```

Query Loop

You can loop over the results of a ColdFusion query.

```
<cfquery name="getMovies" datasource="Entertainment">
    select top 4 movieName
    from Movies
</cfquery>
<cfloop query="getMovies">
    #movieName#
</cfloop>
```

List Loop

You can use the (optional) `delimiters` attribute to specify which characters are used as separators in the list.

```
<cfloop list="ColdFusion,HTML;XML" index="ListItem" delimiters=",;">
    <cfoutput>
        #ListItem#<br />
    </cfoutput>
</cfloop>
```

File Loop

You can loop over a file.

```
<cfloop file="#myFile#" index="line">
    <cfoutput>
        #line#<br />
    </cfoutput>
</cfloop>
```

COM Collection/Structure Loops

You can loop over a Structure or COM collection.

```
<cfset myBooks = StructNew()>
<cfset myVariable = StructInsert(myBooks,"ColdFusion","ColdFusion MX Bible")>
<cfset myVariable = StructInsert(myBooks,"HTML","HTML Visual QuickStart")>
<cfset myVariable = StructInsert(myBooks,"XML","Inside XML")>
<cfloop collection="#myBooks#" item="subject">
    <cfoutput>
        <b>#subject#:</b> #StructFind(myBooks,subject)#<br />
    </cfoutput>
</cfloop>
```

Read CFLOOP How-To online: https://riptutorial.com/coldfusion/topic/3035/cfloop-how-to

Chapter 3: cfquery

Parameters

Parameter	Details
name	Value: *string*, Default: yes
dbtype	Value: query/hql, Default: no, Remarks: when left blank, it's a normal query
datasource	Default: no, Remarks: database
params	Value: *structure*, Default: no, Remarks: cfscript syntax only! In cfml they are written inside SLQ stament using `<cfqueryparam />`

Examples

Using cfquery within a Function

```
<cffunction name="getUserById" access="public" returntype="query">
    <cfargument name="userId" type="numeric" required="yes" hint="The ID of the user">

    <cfquery name="local.qryGetUser" datasource="DATABASE_NAME">
         SELECT  id,
                 name
         FROM    user
         WHERE   id = <cfqueryparam value="#arguments.userId#" cfsqltype="cf_sql_integer">
    </cfquery>

    <cfreturn local.qryGetUser>
</cffunction>
```

Query of Query

Function Calls

```
<!--- Load the user object based on the component path. --->
<cfset local.user = new com.User() />
<cfset local.allUsers = user.getAllUsers()>
<cfset local.specificUser = user.getUserIdFromQry(qry = local.allUsers, userId = 1)>
```

User.cfc

```
<cfcomponent>
    <cffunction name="getAllUsers" access="public" returntype="query">
        <cfquery name="local.qryGetAllUsers" datasource="DATABASE_NAME">
```

```
            SELECT  id,
                    name
            FROM    user
        </cfquery>

        <cfreturn local.qryGetAllUsers>
    </cffunction>

    <cffunction name="getUserIdFromQry" access="public" returntype="query">
        <cfargument name="qry" type="query" required="Yes" hint="Query to fetch from">
        <cfargument name="userId" type="numeric" required="Yes" hint="The ID of the user">

        <cfquery name="local.qryGetUserIdFromQry" dbtype="query">
            SELECT  id,
                    name
            FROM    arguments.qry
            WHERE   id = <cfqueryparam value="#arguments.userId#" cfsqltype="cf_sql_integer">
        </cfquery>

        <cfreturn local.qryGetUserIdFromQry>
    </cffunction>
</component>
```

Read cfquery online: https://riptutorial.com/coldfusion/topic/6452/cfquery

Chapter 4: ColdFusion Arrays

Syntax

- ArrayNew(dimension, isSynchronized)

Parameters

Name	Description
Dimension	Number of dimensions in new array: 1, 2, or 3
isSynchronized	When *false*, creates an unsynchronized array, When *true*, the function returns a synchronized array.

Remarks

In a synchronized array, more than two threads cannot access the array simultaneously. Other threads has to wait until the active thread completes its job, resulting in significant performance.

In 2016 ColdFusion release, you can use an unsynchronized array and let multiple threads access the same array object simultaneously.

Examples

Creating Arrays

Creating arrays explicitly using ArrayNew()

Declare an array with the ArrayNew function. Specify the number of dimensions as an argument.

- ArrayNew(*dimension*) creates an array of 1–3 dimensions.
- ColdFusion arrays expand dynamically as data is added.
- ArrayNew() returns an array.

History

Introduced in ColdFusion MX 6

Declaration

CFML

```
<!--- One Dimension Array--->
<cfset oneDimensionArray = ArrayNew(1)>
```

CFScript Note that inside a function you should use `var` scoping. Earlier versions of CF required var scoping to be the first thing in a function; later versions allow it anywhere in a function.

```
<cfscript>
    oneDimensionArray = ArrayNew(1);

    public void function myFunc() {
        var oneDimensionArray = ArrayNew(1);
    }
</cfscript>
```

After creating the array, add elements by using the element indexes. The Coldfusion Array index starts from 1:

CFML

```
<cfset oneDimensionArray[1] = 1>
<cfset oneDimensionArray[2] = 'one'>
<cfset oneDimensionArray[3] = '1'>
```

CFScript

```
<cfscript>
    oneDimensionArray[1] = 1;
    oneDimensionArray[2] = 'one';
    oneDimensionArray[3] = '1';
</cfscript>
```

Using ArrayAppend()

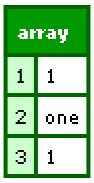
You can add elements to an array using the function `ArrayAppend(array, value)`.

```
<cfscript>
    ArrayAppend(oneDimensionArray, 1);
    ArrayAppend(oneDimensionArray, 'one');
    ArrayAppend(oneDimensionArray, '1');
</cfscript>
```

Output the array contents using `<cfdump>`:

```
<cfdump var="#oneDimensionArray#">
```

Results:



```
array
1  1
2  one
3  1
```

CFML

```
<!--- Two Dimension Array--->
<cfset twoDimensionArray = ArrayNew(2)>
<cfset twoDimensionArray[1][1] = 1>
<cfset twoDimensionArray[1][2] = 2>
<cfset twoDimensionArray[2][1] = 3>
<cfset twoDimensionArray[2][2] = 4>
```

CFScript

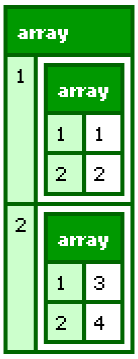
```
<cfscript>
    twoDimensionArray = ArrayNew(2);

    twoDimensionArray[1][1] = 1;
    twoDimensionArray[1][2] = 2;
    twoDimensionArray[2][1] = 3;
    twoDimensionArray[2][2] = 4;
</cfscript>
```

Outputting the contents of array using `<cfdump>`

```
<cfdump var="#twoDimensionArray#">
```

Result:



```
array
1  array
   1  1
   2  2
2  array
   1  3
   2  4
```

Each element contains another Array, which will store the values.

Creating 1-D Array Implicitly

When creating an array implicitly, brackets ([]) surround the array contents with comma separators.

```
<cfset oneDimensionArrayImplicit = [ 1 ,'one','1' ]>
```

This statement is equivalent to the four statements used to create the above oneDimensionArray. The result are the same when using:

```
<cfdump var="#oneDimensionArrayImplicit#">
```

Create 2-D Array Implicitly

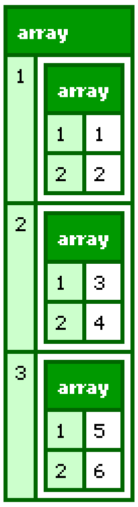
```
<cfset twoDimensionArrayImplicit = [[ 1 , 2 ],[ 3 , 4 ],[ 5 , 6 ]]>
```

Or:

```
<cfset firstElement = ["1", "2"]>
<cfset secondElement= ["3", "4"]>
<cfset thirdElement = ["5", "6"]>
<cfset twoDimensionArrayImplicit = [firstElement , secondElement, thirdElement]>
```

Outputting the content using

```
<cfdump var="#twoDimensionArrayImplicit#">
```



```
array
1  array
   1  1
   2  2
2  array
   1  3
   2  4
3  array
   1  5
   2  6
```

Alternative Explicit Declaration

Also you can declare 1 Dimension Array as

```
<cfset oneDimensionArray = []>

<cfscript>
    oneDimensionArray = [];
</cfscript>
```

This declaration is same as that of using `ArrayNew(1)`.

But if you try declaring 2 Dimension Array as

```
<cfset twoDimensionArray =[][]>    //Invalid CFML construct
```

an error will occur while processing this request.

Following statement will process the request:

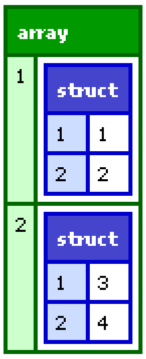
```
<cfset twoDimensionArray =[]>
```

but variable `twoDimensionArray` will not actually an Array within Array (or 2-Dimension Array). It actually contains structure within Array.

```
<cfset twoDimensionArray =[]>
<cfset twoDimensionArray[1][1] = 1>
<cfset twoDimensionArray[1][2] = 2>
<cfset twoDimensionArray[2][1] = 3>
<cfset twoDimensionArray[2][2] = 4>

<cfdump var="#twoDimensionArray#">
```

Result:



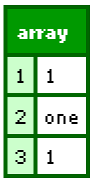
```
array
1  struct
   1  1
   2  2
2  struct
   1  3
   2  4
```

Array in CFScript

```
<cfscript>
    oneDimensionArray = ArrayNew(1);
    oneDimensionArray[1] = 1;
    oneDimensionArray[2] = 'one';
    oneDimensionArray[3] = '1';
</cfscript>

<cfif IsDefined("oneDimensionArray")>
    <cfdump var="#oneDimensionArray#">
</cfif>
```

Result:



```
array
1  1
2  one
3  1
```

Also, we can declare an one Dimension Array as:

```
oneDimensionArray = [];
```

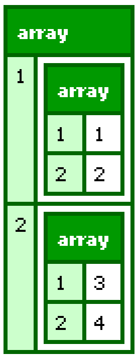
Alternatively, CF introduced `WriteDump()` from **CF9** as a function equivalent to the `<cfdump>` tag which can be used in `<cfscript>`.

```
<cfscript>
    WriteDump(oneDimensionArray);
</cfscript>
```

Similarly, for 2 Dimension Array:

```
<cfscript>
    twoDimensionArray = ArrayNew(2);
    twoDimensionArray[1][1] = 1;
    twoDimensionArray[1][2] = 2;
    twoDimensionArray[2][1] = 3;
    twoDimensionArray[2][2] = 4;
</cfscript>
<cfdump var="#twoDimensionArray#">
```

Result:



```
array
1  array
   1  1
   2  2
2  array
   1  3
   2  4
```

General information

First some general information about how arrays behave in Coldfusion as compared to other programming languages.

- Arrays can have numeric indexes only (if you want to have a string index use `struct`s instead)
- Arrays begin at index [1]
- Arrays can have one ore more *dimensions*

Read ColdFusion Arrays online: https://riptutorial.com/coldfusion/topic/6896/coldfusion-arrays

Chapter 5: Creating REST APIs in coldfusion

Introduction

REST APIs are interesting when data should be accessed from everywhere including different languages (server and client side). That requires separation from data and processing.

Examples

Creating backend

```
<cfcomponent displayname="myAPI" output="false">
    <cffunction name="init" access="public" output="no">
        <!--- do some basic stuff --->
        <cfreturn this>
    </cffunction>

    <cffunction name="welcome">
        <cfreturn "Hello World!">
    </cffunction>
</cfcomponent>
```

The interface

```
<cfscript>
    api_request = GetHttpRequestData();
    api = createObject("component","myAPI").init();
</cfscript>

<cfif api_request.method is 'GET'>
    <cfoutput>#api.welcome()#</cfoutput>
<cfelseif api_request.method is 'POST'>
    <cfheader statuscode="500" statustext="Internal Server Error" />
</cfif>
```

Read Creating REST APIs in coldfusion online:
https://riptutorial.com/coldfusion/topic/10698/creating-rest-apis-in-coldfusion

Chapter 6: Database Queries

Examples

Working with databases

One of ColdFusion's strengths is how easy it is to work with databases. And of course, query inputs can and should be parameterized.

Tag Implementation

```
<cfquery name="myQuery" datasource="myDatasource" result="myResult">
    select firstName, lastName
    from users
    where lastName = <cfqueryparam value="Allaire" cfsqltype="cf_sql_varchar">
</cfquery>
```

CFScript Implementation

```
// ColdFusion 9+
var queryService = new query(name="myQuery", datasource="myDatasource");
queryService.addParam(name="lName", value="Allaire", cfsqltype="cf_sql_varchar");
var result = queryService.execute(sql="select firstName, lastName from users where lastName =
:lName");
var myQuery = result.getResult();
var myResult = result.getPrefix();

// ColdFusion 11+
var queryParams = {lName = {value="Allaire", cfsqltype="cf_sql_varchar"}};
var queryOptions = {datasource="myDatasource", result="myResult"};
var myQuery = queryExecute("select firstName, lastName from users", queryParams,
queryOptions);
```

Inserting values is just as easy:

```
queryExecute("
    insert into user( firstname, lastname )
    values( :firstname, :lastname )
",{
    firstname: { cfsqltype: "cf_sql_varchar", value: "Dwayne" }
    ,lastname: { cfsqltype: "cf_sql_varchar", value: "Camacho" }
},{
    result: "local.insertResult"
});

return local.insertResult.generated_key;
```

Basic Example

Database connections are set up using the CF Administrator tool. See Database Connections for how to connect a datasource.

To execute queries all you need is the `<cfquery>` tag. The `<cfquery>` tag connects to and opens the database for you, all you need to do is supply it with the name of the datasource.

```
<cfquery name="Movies" datasource="Entertainment">
    SELECT title
    FROM   Movies
</cfquery>
```

To display the query results:

```
<cfoutput query="Movies">
    #title#<BR>
</cfoutput>
```

Authentication

Many database configurations require authentication (in the form of a username and password) before you can query the database. You can supply these using the username and password attributes.

Note: the username and password can also be configured against the datasource in the ColdFusion Administrator. Supplying these details in your query overrides the username and password in the ColdFusion Administrator.

```
<cfquery datasource="Entertainment" username="webuser" password="letmein">
    select *
    from Movies
</cfquery>
```

Cached Queries

A cached query is a query that has its results stored in the server's memory. The results are stored when the query is first run. From then on, whenever that query is requested again, ColdFusion will retrieve the results from memory.

You can cache a query using the `cachedAfter` attribute. If the query was last run after the supplied date, cached data is used. Otherwise the query is re-run.

```
<cfquery datasource="Entertainment" cachedAfter="July 20, 2016">
    select *
    from Movies
</cfquery>
```

In order for the cache to be used, and multiple calls to the database be avoided the current query must use the same SQL statement, data source, query name, user name, and password as the cached query used. This includes whitespace in the query.

As such the following queries create different caches, even though the trimmed characters are the same and the query results are identical:

```
<cfquery datasource="Entertainment" cachedAfter="July 20, 2016">
    select *
    from Movies
    <cfif false>
    where 1 = 1
    </cfif>
    <cfif true>
    where 1 = 1
    </cfif>
</cfquery>

<cfquery datasource="Entertainment" cachedAfter="July 20, 2016">
    select *
    from Movies
    <cfif true>
    where 1 = 1
    </cfif>
    <cfif false>
    where 1 = 1
    </cfif>
</cfquery>
```

Limiting the Number of Records Returned

You can limit the number of rows to be returned by using the `maxrows` attribute.

```
<cfquery datasource="Entertainment" maxrows="50">
    select *
    from Movies
</cfquery>
```

Timeouts

You can set a timeout limit using the `timeout` attribute. This can be useful in preventing requests running far longer than they should and impacting on the whole application as a result.

The `timeout` attribute sets the maximum number of seconds that each action of a query is allowed to execute before returning an error.

```
<cfquery datasource="Entertainment" timeout="30">
    select *
    from Movies
</cfquery>
```

Read Database Queries online: https://riptutorial.com/coldfusion/topic/4582/database-queries

Chapter 7: How to invoke a private method dynamically

Remarks

Use of `<cfinvoke>` or `invoke()` should be faster than `evaluate()`

Examples

CFML

```
<cfinvoke method="#somePrivateMethodName#">
  <cfinvokeargument name="argument1" value="one">
</cfinvoke>
```

CFSCRIPT (CF10+)

```
invoke("", somePrivateMethodName, {argument1='one'});
```

Read How to invoke a private method dynamically online:
https://riptutorial.com/coldfusion/topic/6110/how-to-invoke-a-private-method-dynamically

Chapter 8: Scopes in Coldfusion

Introduction

A **scope** is "the range in which a variable can be referenced". ColdFusion knows — as well as most other programming and script languages — several scopes. The following text deals with these types and trys to bring clarity about them, their differences and their characteristics.

Examples

Request Scopes

request

variables

form

url

cgi

Global Scopes

Server

Application

Session

Components and functions

variables

this

local

arguments

Custom tags

attributes

thisTag

caller

Common scopes

Mostly you're probably working with these scopes:

- **Variables scope**is the scope where all variables are assigned to when nothing else is intentionally declared (like the `window` scope in JavaScript).
- **Form scope**When you send a form to your server, all the form fields which can be identified (by setting the name/id property) are accessible in this scope for further server-side processing.
- **URL scope**All url query params are stored in that scope
- **this scope**Inside a component the `this` refers to the component itself
- **local scope**Variables declared inside a function using the `local` statement are encapsulated and only accessible inside that specific function (this is made to avoid pollution of other sopes)
- **Arguments scope**Arguments passed to a function inside a component declared by the `cfargument` tag are accessible with that scope

Overview

- Components and functions
    - variables
    - this
    - local
    - arguments
- Custom tags
    - attributes
    - thisTag
    - caller
- Global Scopes
    - Server
    - Application
    - Session
- Request Scopes
    - request
    - variables
    - form
    - url
    - cgi

Read Scopes in Coldfusion online: https://riptutorial.com/coldfusion/topic/7864/scopes-in-coldfusion

Chapter 9: Variables

Parameters

Attribute	Description
name	(Required) Name of the parameter/variable.
default	(Optional) Value to set parameter to if it does not exist.
max	(Optional) The maximum valid value; used only for range validation.
min	(Optional) The minimum valid value; used only for range validation.
pattern	(Optional) A JavaScript regular expression that the parameter must match; used only for regex or regular_expression validation.
type	(Optional) The valid format for the data.

Examples

Using cfset

You can set a ColdFusion variable using the `<cfset>` tag. To output the variable, you need to surround the variable name with hash `#` symbols and enclose it within `<cfoutput>` tags.

```
<cfset variablename="World!">
<cfoutput>
    Hello #variablename#
</cfoutput>
```

Using cfparam

The `<cfparam>` tag creates a variable if it does not already exist. You can assign a default value using the `default` attribute. This can be used if you want to create a variable, but don't want to overwrite it if it has been previously created elsewhere.

Here the variable hasn't been set previously, so it will be assigned with the `<cfparam>` tag.

```
<cfparam name="firstName" default="Justin">
<cfoutput>
    Hello #firstName#
</cfoutput>
```

Here the variable has already been assigned using the `<cfset>` tag, so this value will override the default value in the `<cfparam>` tag.

```
<cfset firstname="Justin">

<cfparam name="firstName" default="Barney">
<cfoutput>
    Hello #firstName#
</cfoutput>
```

Checking if a Variable Exists

You can check if a variable has been defined in a scope by using ColdFusion's built in `StructKeyExists()` function. This can be used inside a `<cfif>` tag to prevent error messages in the event you attempt to refer to a variable that does not exist. You can also use this function to determine whether a user has performed a certain action or not. The syntax for the function is

```
StructKeyExists(structure, "key")
```

The following example checks if the variable `firstName` exists in the `variables` scope.

```
<cfif StructKeyExists(variables, "firstName")>
    Hello #variables.firstname#!
<cfelse>
    Hello stranger!
</cfif>
```

Alternatively, you may use the function:

```
isDefined("scopeName.varName")
```

To avoid ambiguity, it is recommended to declare the scope. For example, If you have a variable in the scope `test`

```
<cfset test.name = "Tracy" />
```

and you test for `name` in the global scope, you will get a result of `true`.

```
isDefined("name") <!--- true --->
isDefined("x.name") <!--- false--->
isDefined("test.name") <!--- true --->
```

Setting a variable scope

It is a common practice to set application variables to an object scope. This keeps them easy to identify and distinguish from variables in other scopes.

The Variables scope in a CFC is private to the CFC. When you set variables in this scope, they cannot be seen by pages that invoke the CFC.

```
<cfparam name="variables.firstName" default="Timmy">
<cfset variables.firstName="Justin">
```

Scopes shared with the calling page include: Form, URL, Request, CGI, Cookie, Client, Session, Application, Server, and Flash. Variables in these scopes are also available to all pages that are included by a CFC.

CFC:

```
<cfset url.sessionId="23b5ly17">

<cfinclude template="check_session.cfm">
```

check_session.cfm

```
<cfif url.sessionId eq "23b5ly17">
    <p>Welcome back!</p>
</cfif>
```

Read Variables online: https://riptutorial.com/coldfusion/topic/4904/variables

Chapter 10: Working with RegExp Replace callbacks

Introduction

If you want more than a simple string replacement with common regular expressions you certainly run into trouble and hit the wall when discovering the limits of the regex functions Coldfusion has. There is no build-in function like php's `preg_replace_callback`.

Parameters

Parameter	Details
`re`	The regular expression
`str`	The string which should be applyed the the regex
`callback`	The function where the captured grouped will be passed in if a match was found. There the matches can be processed

Remarks

Because Coldfusion itself does not offer what we want, we make recourse to the variety of Java, which is — as we all know — on top of Coldfusion. Java offers us `java.util.regex.Pattern`.

So here is what we actually do:

1. Invoke the `Compile` method from the `Pattern` Class object and passing the regex pattern to it (which probably deposits the regex pattern for later use).
2. Invoke the `Matcher` method on what the `Compile` method returned and passing the string where the pattern should be executed.
3. Test if matching was successfull by invoking the `find` method on what the `Matcher` method returned.

If `matcher.find()` returns `true`, we could say "That's it", but there is one little thing we have to consider: Java's Pattern object stores the groups and gives us access via another function, which is not always the best way for further processing and not that consistent regarding how other programming languages handle this case. Therefore we loop over `matcher.group()` so that we can pass an array containing the captured groups to the callback function. And now we can say: "That's it!"

Examples

User defined REReplaceCallback function

```
function REReplaceCallback(re,str,callback) {
    /*
        Thanks to Ben Nadel
        "Learning ColdFusion 8: REMatch() For Regular Expression Matching"
        from 2007-06-13
        https://www.bennadel.com/blog/769-learning-coldfusion-8-rematch-for-regular-
expression-matching.htm
    */
    pattern = CreateObject("java","java.util.regex.Pattern").Compile(Arguments.re);
    matcher = pattern.Matcher(Arguments.str);
    if(matcher.find()) {
        groups = [];
        for(var i = 1; i lte matcher.groupCount(); i++) {
            ArrayAppend(groups,matcher.group(Val(i)));
        }
        return Arguments.callback(groups);
    }
    else {
        return Arguments.callback(false);
    }
}
```

Using REReplaceCallback function

```
REReplaceCallback('YOUR REGEX GOES HERE','AND YOUR STRING HERE',function(groups) {
    //now you can access the 'groups' array containing all the captured groups
    return result; //return whatever you've processed inside
});
```

Read Working with RegExp Replace callbacks online:
https://riptutorial.com/coldfusion/topic/10655/working-with-regexp-replace-callbacks

Credits

S. No	Chapters	Contributors
1	Getting started with coldfusion	Adam Tuttle, Adrian J. Moreno, Community, Justin Duhaime, mhatch, RamenChef, shaedrich, Steven Benjamin, user3071284, William Giles
2	CFLOOP How-To	Adrian J. Moreno, Bonanza, mhatch, RRK
3	cfquery	Bubblesphere, shaedrich
4	ColdFusion Arrays	4444, Justin Duhaime, mhatch, Mishra Shreyanshu, shaedrich
5	Creating REST APIs in coldfusion	shaedrich
6	Database Queries	Adam Tuttle, Leigh, mhatch, nosilleg, shaedrich, user3071284
7	How to invoke a private method dynamically	Henry
8	Scopes in Coldfusion	James A Mohler, shaedrich
9	Variables	mhatch
10	Working with RegExp Replace callbacks	shaedrich