Arrays

Colin Walker, 2008-05-05

What is an array? Wikipedia would tell you - "In computer science an array is a data structure consisting of a group of elements that are accessed by indexing. In most programming languages each element has the same data type and the array occupies a contiguous area of storage." - which is correct, but still a bit unclear. An array is, quite simply, a memory structure (variable) that allows you to store more than one value. Usually these values are associated somehow logically.

For instance, suppose you need to store and retrieve 10 different IP addresses. Which would be easier, having 10 different variables, presumably $ipAddr1 - $ipAddr10, or a single variable with multiple values associated - $ipAddr[0] - $ipAddr[9]? This is exactly what an array does, and one of the reasons you might use one. Arrays are also one of the fastest, most efficient ways to store large numbers of strings in an iRule. This can prove extremely useful in scenarios where you're doing things like IP based connection limiting or the like, where you need to store a vast number of strings, and associate another piece of data with them.

Arrays are made up of three parts, the array name, which references the structure itself, the indexes, which are the values used as pointers to reference the data stored in relation to those indexes, and the values, which are referenced via the indexes, and are usually the data you're trying to store or retrieve. To clarify:

```
array set colorcount {   # colorcount is the array name for this example
   green 5                # On this line, green represents an index
   blue  4               # Here the number 4 represents a value.
}                        # If I wanted to later find out how many green items were stored in this a
                         # I could retrieve that value by calling the array name with the appropria
                         # like this: $colorcount(green), which would return the associated value,
```

There is plenty of documentation out there about arrays and how they work, but we want to focus specifically on the use of arrays in iRules, and as such arrays in TCL. Arrays in TCL happen to be a little bit different than the average structure, so let's clarify a bit more. TCL arrays are actually hash tables and don't actually have anything in common with the data structures called arrays in many other programming languages. A TCL array provides a rapid answer to the question "is there a value associated with this key". Arrays in most languages can only be indexed by integers (Meaning $ipAddr[1], $ipAddr[2]) wheras arrays in TCL are really "Associative Arrays", meaning they, like nearly everything else in TCL, are built based off of strings. Because of this, it means that you can use any string value as an index in your array, rather than just integers (I.E. $ipAddr[1], $ipAddr[last], $ipAddr[first], $ipAddr[$host]). This is why we're able to use green and blue as indexes in the above example, rather than being forced to use an integer to index the values.

TCL Arrays have many functions that go along with them. There are array functions to set, get, remove, update, search and much more. Many of these can be extremely useful. We'll cover a few of them here. For more complete documentation I strongly recommend checking out the sourceforge TCL array documentation.

First let's look at array set:

> **array set** arrayName list
> Sets the values of one or more elements in arrayName. list must have a form like that returned by array get, consisting of an even number of elements. Each odd-numbered element in list is treated as an element name within arrayName, and the following element in list is used as a new value for that array element. If the variable arrayName does not already exist and list is empty, arrayName is created with an empty array value.

We've already gone over a simple array set example in the code above. This shows how you would create a simple array by manually setting the indexes and values. Keep in mind, though, that these values can also be set in a more dynamic fashion by referencing other variables, such as:

```
array set hosts {
  [HTTP::host] 1
}
```

Some of the other functions you'll definitely want to know about are:

> **array get** arrayName ?pattern?
> Returns a list containing pairs of elements. The first element in each pair is the name of an element in arrayName and the second element of each pair is the value of the array element. The order of the pairs is undefined. If pattern is not specified, then all of the elements of the array are included in the result. If pattern is specified, then only those elements whose names match pattern (using the matching rules of string match) are included. If arrayName is not the name of an array variable, or if the array contains no elements, then an empty list is returned. If traces on the array modify the list of elements, the elements returned are those that exist both before and after the call to array get.
>
> **array size** arrayName
> Returns a decimal string giving the number of elements in the array. If arrayName is not the name of an array then 0 is returned.
>
> **array names** arrayName ?mode? ?pattern?
> Returns a list containing the names of all of the elements in the array that match pattern. Mode may be one of -exact, -glob, or -regexp. If specified, mode designates which matching rules to use to match pattern against the names of the elements in the array. If not specified, mode defaults to -glob. See the documentation for string match for information on glob style matching, and the documentation for regexp for information on regexp matching. If pattern is omitted then the command returns all of the element names in the array. If there are no (matching) elements in the array, or if arrayName is not the name of an array variable, then an empty string is returned.

These can be used individually or together to create some pretty powerful logic dealing with the array structures you're using to store your data. Whether you're looping through the array based on the list of names, or searching for certain index patterns, these commands open up a host of possibilities for things that you can do with arrays. Here are a few more examples:

A simple example of looping through the above array "colorcount" using the array names command:

```
foreach color [array names colorcount] {
   log "Color: $color Count: $colorcount($color)"
}

Results in -
    Color: blue Count: 4
    Color: green Count: 5
```

Another foreach loop, this time using the array get command and some more advanced logic dealing with the results. The key difference here is that the element value is being returned with the array get command, and just the element index is being returned with the above array names command. Choosing which one you use will depend on what you're trying to accomplish.

```
array set ersetzer {
  "https://portal.domain:80" "https://portal.domain.net"
  "http://portal" "https://portal"
}

foreach {myfind myreplace} [array get ersetzer] {
  set offset 0
  set diff [expr [string length $myreplace] - [string length $myfind]]
  # Get indices of all instances of find string in the payload
  set indices [regexp -all -inline -indices $myfind [HTTP::payload]]
  ...
}
```

It's important to note, also, that aside from the many functions that go along with arrays for manipulation, you can read from and write to them directly, like a normal variable, provided you use the proper syntax. There is a special notation for array names that allows you to directly access them for storing or retrieving data:

```
    set a(1) 0
    set salary(Smith) 30000
    set age(Smith) 45
    set length($s) [string length $s]
    incr ipAddr([IP::local_addr])
    log "Smith's salary = $salary(Smith)"
    log "[IP::local_addr] has $ipAddr([IP::local_addr]) connections open"
```

As you can see there are many different uses for arrays in your iRule. I invite you to explore the use of these commands and this data type, as it is very fast, useful, and relatively straight-forward to use, once you get the hang of it. For further examples check out the DevCentral codeshare or simply search the site for "array".

Get the Flash Player to see this player.