iRules Pointer Tables Demystified

Colin Walker, 2010-13-09

Last week I talked about Heatmaps and some of the really fun things you can do with them via iRules. In that article I covered the concept of pointer tables somewhat briefly and I wanted to spend a little more time discussing it here to make sure that it has a place to live by itself, not just under the umbrella of heatmaps, since it's a very valuable technique in and of itself.

When you're storing things in most memory structures you're using a name value pairing. A variable uses the variable as the name, and the value as, well, the value. In a table it's often times a table name, and then a list of name->value pairs inside that table. This is normal and effective, and keeps data nicely organized. What sometimes happens though is that you'll want to be able to go deeper than just two layers. For instance, you're tracking the sandwiches people want for lunch because you know, you need to build a table structure in code for that…just bear with me. It would look something like this:

Sandwiches		
	Bob	Turkey
	Sally	Ham
	John	Turkey

To get the info you want, you'd just use a simple loop and go through the sandwiches table and see what everyone wants. You'd do something like:

```
1: foreach person [table keys -subtable sandwiches] {
2:    # Do whatever you want with the person here
3:    # To retrieve which sandwich $person wants, do a lookup:
4:    set sandwich [table lookup -subtable sandwiches $person]
5: }
```

You'd end up with Bob->Turkey, Sally->Ham and John->Turkey as your name->value pairs, which is exactly what you were looking for. That's great, assuming they all want the same sandwich every single day. What if they want to change things based on the day? What you now need is something that will store a third level of data, like this:

Sandwiches			
	Monday		
		Bob	Turkey
		Sally	Ham
		John	Turkey
	Tuesday		
		Bob	Turkey
		Sally	Veggie
		John	Roast Beef

This would allow you to nest two loops like the one above to get the same information as before, but now sorted by which day of the week they want said sandwich on. That would be great and all, but here's the rub: you can't do that in iRules. You can't create tables two layers deep, which means we need to figure out another way to deal with the same data.

The way I did it in the heatmaps rule was to create what's often called a pointer table. The idea is straight-forward: Build up, not down. What the heck do I mean by that? Well, we can't create more layers of depth in the table itself, so create more tables, and create a list of those tables, and iterate through multiple tables rather than just one.

So rather than the above structure which is all neat and stored in one table, we'd essentially have:

mytables		
	Monday	
	Tuesday	
	Wednesday	
	Thursday	
	Friday	
Monday		
	Bob	Turkey
	Sally	Ham
	John	Turkey
Tuesday		
	Bob	Turkey
	Sally	Veggie
	John	Roast Beef

Keep in mind here that mytables, Monday and Tuesday are all separate tables in their own right, with nothing linking them logically besides the reference in mytables to the names of the other tables where the actual data we want is stored. So, now that we have this loose association, we can loop through the days and find out what sandwiches people want on each day:

```
1: foreach day [table keys -subtable mytables] {
2:   # You now have the name of the day in the $day variable to do with what you wish here, or below
3:   foreach person [table keys -subtable $day] {
4:   # Now you have both the $day and the $person, so you can easily get the value for what sandwich $person wants,
5:   # and have all 3 pieces of data related just the way you wanted.
6:     set sandwich [table lookup -subtable $day $person]
7:     log local0. "$Person wants a $sandwich because it's $day"
8:   }
9: }
```

So, as you can see, not only can you absolutely achieve the functionality we set out to via iRules, namely retrieving 3 pieces of data across multiple logical layers, but with this same concept you could layer things almost indefinitely by continuing to reference table names that reference table names, etc. Keep in mind the more loops the larger the performance hit, and that nested loops in general are expensive, but this can be a pretty powerful solution when called for.

Hopefully that helps demystify the idea of pointer tables a little bit, and continues to show off just how darn powerful the table command in iRules can be. For an example of this logic in action beyond the worthy task of sandwich selection, check out the Heatmaps, iRules Style: Part 3 article from last week.

Technorati Tags: iRules,Pointer Tables,Advanced,Colin Walker

Related Articles

- F5 DevCentral > Hot Topics > iRules
- Heatmaps, iRules Style: Part2 > DevCentral > F5 DevCentral > Tech Tips

- Colin Walker
- Heatmaps, iRules Style: Part 3 – URL Filtering > DevCentral ...
- iRules 101 #17 – Mapping Protocol Fields with the Binary Scan ...
- Lori MacVittie - iRules
- iRules-O-Rama

F5 Networks, Inc. | 401 Elliot Avenue West, Seattle, WA 98119 | 888-882-4447 | f5.com

F5 Networks, Inc.
Corporate Headquarters
info@f5.com

F5 Networks
Asia-Pacific
apacinfo@f5.com

F5 Networks Ltd.
Europe/Middle-East/Africa
emeainfo@f5.com

F5 Networks
Japan K.K.
f5j-info@f5.com