Advanced iRules: Regular Expressions

Jason Rahm, 2016-10-06

A regular expression or regex for short is essentially a string that is used to describe or match a set of strings, according to certain syntax rules.

Regular expressions ("REs") come in two basic flavors: extended REs ("ERE"s) and basic REs ("BREs"). For you unix heads out there, EREs are roughly the same as used by traditional egrep, while BREs are roughly those of the traditional ed. The TCL implementation of REs adds a third flavor, advanced REs ("AREs") which are basically EREs with some significant extensions.

It is beyond the scope of this article to document the regular expression syntax. A great reference is included in the re_syntax document in the TCL Built-In Commands section of the TCL documentation.

Regular Expression Examples

So what does a regular expression look like? It can be as simple as a string of characters to search for an exact match to "abc"

`{abc}`

Or a builtin escape string that searches for all sequences of non-whitespace in a string

`{\S+}`

Or a set of ranges of characters that search for all three lowercase letter combinations

`{[a-z][a-z][a-z]}`

Or even a sequence of numbers representing a credit card number.

`{(?:3[4|7]\d{13})|(?:4\d{15})|(?:5[1-5]\d{14})|(?:6011\d{12})}`

For more information on the syntax, see the re_syntax manual page in the TCL documentation, you won't be sorry.

Commands that support regular expressions

In the TCL language, the following built in commands support regular expressions:

- regexp - Match a regular expression against a string
- regsub - Perform substitutions based on regular expression pattern matching
- lsearch - See if a list contains a particular element
- switch - Evaluate one of several scripts, depending on a given value

iRules also has an operator to make regular expression comparisons in commands like "if", "matchclass" and "findclass"

- matches_regex - Tests if one string matches a regular expression.

Think twice, no three times, about using Regular Expressions

Regular expressions are fairly CPU intensive and in most cases there are faster, more efficient, alternatives available. There are the rare cases, such as the Credit Card scrubber iRule, that would be very difficult to implement with string searches. But, for most other cases, we highly suggest you search for alternate methods. The "switch -glob" and "string match" commands use a "glob style" matching that is a small subset of regular expressions, but allows for wildcards and sets of strings which in most cases will do exactly what you need.

In every case, if you are thinking about using regular expressions to do straight string comparisons, please, please, please, make use of the "equals", "contains", "starts_with", and "ends_with" iRule operators, or the glob matching mentioned above. Not only will they perform significantly faster, they will do the exact same thing.

Here's a comparative example:

```
BAD: if { [regexp {bcd} "abcde"] } {
BAD: if { "abcde" matches_regex "bcd" } {
BETTER: if { [string match "*bcd*" "abcde"] } {
BEST: if { "abcde" contains "bcd" } {
```

A Performance Challenge: Scan vs Regex

The `scan` and `string` commands will cover the majority of regex use cases, and as indicated above, will save significantly on system resources. Consider the following test where we use Tcl's time command to run a `scan` against an IP address 10,000 times, and then do that again only using the `regexp command`.

```
% set ip "10.10.20.200"
10.10.20.200
% time { scan $ip {%d.%d.%d.%d} a b c d} 10000
2.1713 microseconds per iteration
% time {regexp {([0-9]{1,3})\.([0-9]{1,3})\.([0-9]{1,3})\.([0-9]{1,3})} $ip matched a b c d} 10000
34.2604 microseconds per iteration
```

Two approaches, same result. The time to achieve that result? The `scan` command bests `regexp` by far. I'll save you the calculation…that's a 93.7% reduction in processing time. **93.7 percent**! Now, mind you, the difference between 2 and 34 microseconds will be negligible to an individual request's response time, but in the context of a single system handling hundreds of thousands or even millions of request per second, the difference matters. A lot.

Conclusion

Regular expressions are available for those tricky situations where you just need to perform some crazy insane search like "string contains 123 but only if follows by "abc" or "def", or it contains a 5 digit number that is a valid US zip code". Also, if you need to find the exact location of a string within a string (for instance to replace one string for another), then `regexp` and `regsub` will likely work for you (if a Stream Profile doesn't). So, in some cases, a regular expression is likely the only option. Just keep in mind that even multiple string and comparison tests are more efficient than the most simple regular expression, so use them wisely.