#The101: Events and Priorities

JRahm, 2012-05-09

As #The101: iRules steams on we go deeper and deeper into what actually drives iRules as a technology. So far we have covered very basic concepts, from core programming ideas and F5 basic terminology through to what makes iRules unique and useful, when you'd make use of them, etc. This is all-important as a base of knowledge from which we can draw when speaking about the technology, but now it's time to dig in deeper and start discussing the inner workings of iRules and what makes them tick. If you haven't yet read the first three primer style introduction installments of this article series it is likely worth your time to at least peruse them to be sure you're on the same page before delving into the rest of the series. So far we have:

Introduction to Programming & TCL

Introduction to F5 Technology & Terms

Introduction to iRules

In this installment of #The101: iRules we'll cover two core concepts within iRules, **Events** and **Priorities**. While we will eventually get to commands and different fun tricks that you can do with iRules, sure enough, it's important to understand how iRules actually work first. I can think of no better starting place than events and our built in eventing structure.

Events

What is an event?

iRules is an event driven language. What this means is that there is no freestanding code to be executed each time the script body as a whole is called. Rather code is placed under events, which act like containers, categorizing and separating the code within an iRule into logical sections. This is important to allow iRules to be truly network aware.

When programming in a traditional scripting environment it is normal for the entirety of a script to be executed each time it is called. In a network based scripting environment however you ideally want sections of code to be called at specific moments within the connection flow. I.E. you want to look up an HTTP host name, want to do it after the connection is established, the handshake is performed and the appropriate headers are sent that contain the information you're looking for, but before that request is sent off to the destination server. This means you have to have some way to understand and describe, in your code, that moment in time. There are many similar moments, such as when a connection is first established, when a server is selected for load balancing, when the connection is sent to that server, when a server responds, and so on.

It is because of this that iRules is designed as an event driven language. Thanks to the flexibility of Tcl and our ability to easily modify it to our needs, we were able to extend it to understand a vast number of these moments within the session flow. It is those moments that we refer to as events. Rather than reading every packet that comes across the wire and waiting until you see the information that depicts the HTTP request has finally come across from the user, an iRules user can simply call the HTTP_REQUEST event and trust that the BIG-IP will execute the code contained within that event block at the appropriate moment.

So, very simply, an "Event" when speaking about iRules is a Tcl command that we've added to iRules to describe specific moments within the session flow of a network connection, such as "CLIENT_ACCEPTED" and "HTTP_REQUEST" below.

```
1: when CLIENT_ACCEPTED {
2:    if { [IP::client_addr] eq "192.168.1.1" } {
3:      drop
4:    }
5: }
6:
7: when HTTP_REQUEST {
8:    log local0. "Host: [HTTP::host], Client IP: [IP::client addr]"
```

```
9: }
```

What are the benefits of an event driven language?

In addition to being able to accurately execute code at the specific time desired, which is important considering that often times information will be needed that is only available at specific points in the flow, an event driven language is also a large benefit to performance.

In traditional programming when you make a call to a script the entire script body is executed. This means that if you have a 100 line script written in say Perl or Ruby (nothing against those languages, merely an example), every time that script is called all 100 lines will be executed, generally speaking. This is not true in iRules. With an iRule that is 100 lines long, but spread across 3 or 4 different events, only the specific code applied to each individual event is executed when that event occurs, rather than the entirety of the script.

For instance, if you have a 100 line iRule with 50 lines in a request event and 50 lines in a response event, your BIG-IP is only executing 50 lines of code for every inbound request, rather than having to execute the entire rule. This may not seem like a massive savings, but keep in mind that everything when talking about network side scripting performance becomes more important than in traditional environments, because of the scale generally being discussed. While a traditional script may be executed several hundred times per second in a busy environment, an iRule may be executed several hundred thousand times per second, in a high traffic deployment.

Another way of looking at this, perhaps, is that events themselves are the containers for the scripts being executed against the network. The iRule is a way of grouping those events into a logical container to express a function or segment of application or business logic that you wish to apply as a single object against your network traffic.

What events are available?

Events are broken down into protocols, with usually several events per supported protocol. Events will always follow a profile within BIG-IP, so you can largely correlate which available protocol profiles are available with which protocols have specific events called out within iRules. The list is wide-ranging and I won't list it here, but it is available on DevCentral here (requires registration) for your perusal.

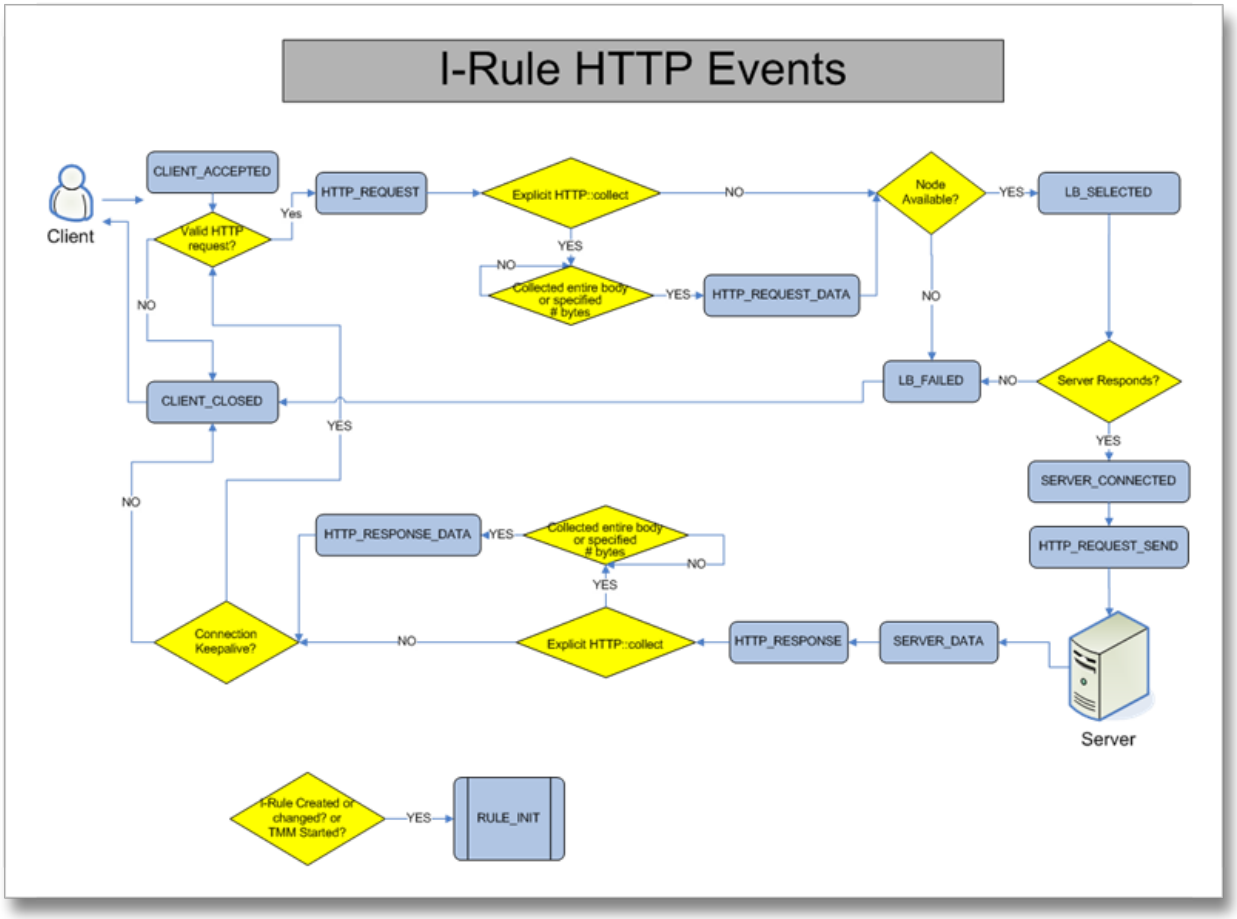
From HTTP to SIP to DNS, most major protocols are represented in the listing and have specific iRules Events, and likely associated commands, designated to them. Keep in mind, however, that even if a particular protocol does not have events specifically set out for it you can accomplish almost anything with base TCP and UDP events and commands. With a strong understanding of iRules and the protocol in question it is entirely possible to parse and manipulate nearly anything passing across the wire with the commands and events already available within the TCP and UDP command sets. It's also worth noting that particular events can be disabled for the duration of the connection with the event disable command. Be cautious when using this, as it will most certainly affect any other iRules applied to the same virtual.

In which order do events fire?

This is a question that gets asked time and time again on DevCentral and out in the field, and it is admittedly a bit difficult to answer. The way that events fire is largely based on the network flow through the proxy within BIG-IP. The proxy architecture within the BIG-IP is made up of a series of filters, each designed to perform specific functions, without getting into tedious detail. Each filter builds upon the last, I.E. the HTTP filter uses information drawn out of the session by the TCP filter, etc. Each of these filters has an associated set of iRules events that fire at that point in the session's life cycle as well, when the connection is being interrogated or affected by the particular filter in question.

As the connection is passed up the chain more and more specific filters fire, along with events to match, until finally the session is handed over from the client side of the proxy to the server side, and it traverses its way down the chain on that side. The return trip for responses follows a similar path with different filters, and as such events, in place; herein lies part of the problem with describing the order of events within an iRule.

There is no static set of events that happen for every single connection. There may be a few specific events that always fire, such as CLIENT_ACCEPTED once per connection, but in general there is no set event structure that fires. The events that fire are dependent on the configuration of the virtual to which the iRule in question is applied. Because of this dynamic nature it is extremely hard to make a blanket statement about which events will fire in a given iRule, let alone event order. Given the entire config of a virtual server it is possible to determine which events will fire, and in which order, but that requires case by case information and analysis. I've provided a diagram of a somewhat generic HTTP iRule to show the logical flow of events, though. This shows the way events relate and flow in a particular configuration's case, and it is a good starting point for discussing event ordering in the case of a standard HTTP virtual.



What happens if I have the same event in multiple iRules on a virtual server?

Many people will have multiple iRules on a single virtual server, some of which will make use of like events. The question of what happens in such a case is often asked, or a user will want to know "Which CLIENT_ACCEPTED will fire first?" If you have a virtual server with multiple iRules all making use of a particular event, there is no "first" or "last" occurrence of that event, it only happens once.

To understand a bit more, first let me explain how iRules are stored and interpreted. iRules are not, for all intents and purposes, a dynamically interpreted language. That is, Tcl interpreter is not spun up every time a request comes into a virtual server with an iRule applied. This would be horribly inefficient. Instead, when an iRule is saved as part of a configuration, it is interpreted by the Tcl engine and pre-compiled down into what is known as "byte code". This is a compiled state that breaks down most of the logic and commands within a given iRule so that at execution time it can very easily and efficiently be fired by the Tcl engine, in this case part of TMM.

During this process the iRule is also effectively broken down and re-organized. The commands that are associated with each event within the iRule are removed from the context of that particular iRule and are instead associated with that event. Remember above where I mentioned the idea of events being tied to a particular filter in the proxy architecture, and that the events fire when those filters are called? This is why that concept is important. The commands you put under your HTTP_REQUEST events are all taken out of whichever iRules they were associated with and instead relegated to live under the event that is responsible for executing those commands.

As such, there is no concept of multiple instances HTTP_REQUEST or CLIENT_ACCEPTED or SIP_RESPONSE or any other event occurring with a given virtual server. The session traffic proceeds up the chain, events fire, and commands are executed in that context, be it client or server. Regardless of how many iRules you have that call a given event, that event only ever fires once per connection/request, depending on the event.

Even if you can't have multiple instances of a given event executed from a particular virtual server, you can still control the order in which your iRules execute. This is done via the next concept we'll cover, priorities.

Priorities

What are priorities?

Priorities are an integer value associated with every iRule that determine the order in which iRules applied to a given virtual server execute, the lower the number, the higher the relative priority. I.E. an iRule with a priority of 1 would fire before an iRule with a priority of 900. If you want to ensure that the variable you're setting in iRuleA exists when iRuleB executes, you'll want to set priorities accordingly.

How do I set priority?

Every iRule has a priority, whether you set one or not. The system will implicitly set a priority value to all iRules so that it can keep straight the order in which to execute commands at run time. This is done automatically when saving iRules without a specified priority.

If, however, you want to set a specific priority, it is done so simply via the priority command. At the beginning of your iRule you can state "priority 100" to designate the priority for that iRule and, by association, all of the commands contained within. If you're going to start setting specific priorities it is recommended to do so for all of your iRules, however, to be sure that the execution order you want is achieved.

```
1: priority 100
2: when HTTP_REQUEST {
3:   if {[HTTP::host] eq "mydomain.com" } {
4:     set host [HTTP::host]
5:   }
6: }
```

Why does priority matter?

In many cases, it doesn't. The vast majority of iRules deployments that I've seen do not have specific priorities set. It can be useful, however, in cases where you want to ensure that a particular order of execution is achieved. A couple of possible examples may be if you're passing particular information from one iRule to another, or want an iRule to act on a modified value of some sort. Whatever the reasons, it is likely that if you aren't already looking for a way to set priority, you don't need to worry about it.

Can I supersede the session flow?

First the simple answer: No. Now the explanation: People sometimes ask if they're able to do things like act on a response before a request, view a load balancing decision as soon as the connection is established, or other things that are unnatural in regards to the flow of a session through the BIG-IP. This is not possible. By their very nature events occur based on the way traffic traverses the proxy, and no amount of priority will change that. You can absolutely ensure that iRuleA's response event occurs before iRuleB's response events, but you cannot force iRuleA's response events to occur before iRuleB's request events.

You can achieve a similar effect with a couple of conditionals and variables, and this is the logic trick that I'd recommend to anyone who needs to mimic this behavior. Keep in mind that this immediately ceases to function effectively when you have multiple iRules on the virtual server, unless all of them are built with this in mind. **Use caution**, as this is absolutely risky. It is not, however, possible to alter the actual event structure or ordering.

```
1: when HTTP_REQUEST {
2:   if { [info exists response fired] } {
```

```
    unset response_fired
    ...
  }
}

when HTTP_RESPONSE {
  set response_fired 1
...
}
```

Next Wednesday, look for the next installment of #The101: iRules, which will cover control structures and operators.

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