链滴

Redis Lua 脚本中学教程（上）

作者：Hawkpool

原文链接：https://ld246.com/article/1584513609755

来源网站：链滴

<img src="https://ld246.com/images/img-loading.svg" alt="" data-src="https://b3logfile
com/bing/20191016.jpg?imageView2/1/w/960/h/540/interlace/1/q/100">
中学教程主要分为两部分：Redis Lua 的相关命令详解和 Lua 的语法介绍。
前面我们简单介绍了 EVAL 和 EVALSHA 命令。但是只有那点只是是没办法从中学毕业的，因此
们需要进行更深入的学习。
<h4 id="EVAL">EVAL</h4>
最早可用版本：2.6.0
用法：<code>EVAL script numkeys key [key ...] arg [arg ...]</code>
关于用法我们已经演示过了，其中第一个参数是要执行的 Lua 脚本，第二个参数是传入脚本的
数个数。后面则是参数的 key 数组和 value 数组。
在 Lua 中执行 Redis 命令的方法我们也介绍过，就是使用 redis.call()和 redis.pcall()两个函数
它们之间唯一的不同就是当 Redis 命令执行错误时，redis.call()会抛出这个错误，使 EVAL 命令抛出
误，而 redis.pcall()会捕获这个错误，并返回 Lua 的错误表。
通常我们约定执行命令的 key 都需要由参数传入，命令必须在执行之前进行分析，以确定它作用
哪个 key。这样做的目的是为了在一定程度上保证 EVAL 执行的 Lua 脚本的正确性。
<h5 id="Lua和Redis之间数据类型的转换">Lua 和 Redis 之间数据类型的转换</h5>
在 Redis 执行 EVAL 命令时，如果脚本中有 call()或者 pcall()命令，就会涉及到 Redis 和 Lua
间数据类型转换的问题。转换规则要求，一个 Redis 的返回值转换成 Lua 数据类型后，再转换成 Redi
数据类型，其结果必须和初始值相同。所以每种类型是一一对应的。转换规则如下：
<h5 id="Redis与Lua互相转换">Redis 与 Lua 互相转换</h5>
<table>
<thead>
<tr>
<th>Redis</th>
<th>Lua</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td>number</td>
</tr>
<tr>
<td>bulk</td>
<td>string</td>
</tr>
<tr>
<td>multi bulk</td>
<td>table</td>
</tr>
<tr>
<td>status</td>
<td>table with a single <code>ok</code> field</td>
</tr>
<tr>
<td>error</td>
<td>table with a single <code>err</code> field</td>
</tr>
<tr>
<td>Nil bulk &Nil multi bulk</td>
<td>false boolean type</td>
</tr>
</tbody>
</table>

除此之外，Lua 到 Redis 的转换还有一些其他的规则：

Lua boolean true -> Redis integer reply with value of 1

Lua 只有一种数字类型，不会区分整数和浮点数。而数字类型只能转换成 Redis 的 integer 类型如果要返回浮点数，那么在 Lua 中就需要返回一个字符串。

Lua 数组在转换成 Redis 类型时，遇到 nil 就停止转换

来个栗子验证一下：
<pre><code class="highlight-chroma">EVAL "return {1,2,3.3333,'foo',nil,'bar'}" 0
1) (integer) 1
2) (integer) 2
3) (integer) 3
4) "foo"
</code></pre>
可以看到 bar 没有返回，并且 3.333 返回了 3。
<h5 id="脚本的原子性">脚本的原子性</h5>
Redis 运行所有的 Lua 命令都使用相同的 Lua 解释器。当一个脚本正在执行时，其他的脚本或 Redis 命令都不能执行。这很像 Redis 的事务 multi/exec。这意味着我们要尽量避免脚本的执行时间过。
<h5 id="脚本整体复制">脚本整体复制</h5>
当脚本进行传播或者写入 AOF 文件时，Redis 通常会将脚本本身进行传播或写入 AOF，而不是用它产生的若干命令。原因很简单，传播整个脚本要比传播一大堆生成的命令的速度要快。
从 Redis3.2 开始，可以只复制影响脚本执行结果的语句，而不用复制整个脚本。这个复制整个本的方法有以下属性：

如果输入相同，脚本必须输出相同的结果。即执行结果不能依赖于隐式的变量，或依赖于 I/O 输

Lua 不会导出访问系统时间或其他外部状态的命令

如果先执行了“随机命令”（如 RANDOMKEY，SRANDMEMBER，TIME），并改变了数据集接着执行脚本时会被阻塞。

在 Redis4 中，Lua 脚本调用返回随机顺序的元素的命令时，会在返回之前进行排序，也就是说调用 redis.call("smembers",KEYS[1])，每次返回的顺序都相同。从 Redis5 开始就不需要排序了，为 Redis5 复制的是产生影响的命令。

Lua 修改了伪随机函数 math.random 和 math.randomseed，使每次执行脚本时 seed 都相同而如果不执行 math.randomseed，只执行 math.random 时，每次的结果也都相同。

<h5 id="复制命令队列">复制命令队列</h5>

在这种模式下，Redis 在执行脚本时会收集所有影响数据集的命令，当脚本执行完毕时，命令队会被放在事务中，发送给 AOF 文件。
Lua 可以通过执行 redis.replicate_commands()函数来检查复制模式，如果返回 true 表示当前复制命令模式，如果返回 false，则是复制整个脚本模式。
<h5 id="可选择的复制命令">可选择的复制命令</h5>
脚本复制模式选择好以后，就可以对复制到副本和 AOF 的方式进行更多的控制。这是一种高级性，因为滥用会切断主从备份，和 AOF 持久化。如果我们只需要在 master 上执行某些命令时，这特性就变得很有用。例如我们需要计算一些中间值时，只需要在 master 上计算就好，那么这些命令不必进行复制。
从 Redis3.2 开始，有一个新的命令叫做 redis.set_repl()，它可以用来控制复制方式，有如下选（默认是 REPL_ALL）：
<pre><code class="highlight-chroma">redis.set_repl(redis.REPL_ALL) -- Replicate to AOF and replicas.
redis.set_repl(redis REPL_AOF) -- Replicate only to AOF.
redis.set_repl(redis REPL_REPLICA) -- Replicate only to replicas (Redis >= 5)
redis.set_repl(redis REPL_SLAVE) -- Used for backward compatibility, the same as REPL_REPLICA.
redis.set_repl(redis REPL_NONE) -- Don't replicate at all.
</code></pre>
<h5 id="全局变量">全局变量</h5>
为了避免数据泄露，Redis 脚本不允许创建全局变量。如果必须有一个公共变量，可以使用 Redis 的 key 来代替。在 EVAL 命令中创建一个全局变量会引起一个异常。
<pre><code class="highlight-chroma">> eval 'a=10' 0
(error) ERR Error r nning script (call to f_933044db579a2f8fd45d8065f04a8d0249383e57): user_script:1: Script at empted to create global variable 'a
</code></pre>
<h5 id="关于SELECT的使用">关于 SELECT 的使用</h5>
在 Lua 脚本中使用 SELECT 就像在正常客户端中使用一样。值得一提的是，在 Redis2.8.12 之，Lua 脚本中执行 SELECT 是会影响到客户端的，而从 2.8.12 开始，Lua 脚本中的 SELECT 只会在本执行过程中生效。这点在 Redis 版本升级时需要注意，因为升级前后，命令的语义会改变。
<h5 id="可用的库">可用的库</h5>
Lua 脚本中有许多库，但并不是都能在 Redis 中使用，其中可以使用的有：

<code>base</code> lib.

<code>table</code> lib.

<code>string</code> lib.

<code>math</code> lib.

<code>struct</code> lib.

<code>cjson</code> lib.

<code>cmsgpack</code> lib.

<code>bitop</code> lib.

<code>redis.sha1hex</code> function.

<code>redis.breakpoint and redis.debug</code> function in the context of the <a href=
https://ld246.com/forward?goto=https%3A%2F%2Flinks.jianshu.com%2Fgo%3Fto%3Dhttps
253A%252F%252Fredis.io%252Ftopics%252Fldb" target="_blank" rel="nofollow ugc">Redis
ua debugger.

struct, CJSON and cmsgpack 是外部库，其他的都是 Lua 的标准库。
<h5 id="在脚本中打印Redis日志">在脚本中打印 Redis 日志</h5>
使用 redis.log(loglevel,message)函数可以在 Lua 脚本中打印 Redis 日志。
loglevel 包括：

redis.LOG_DEBUG

redis.LOG_VERBOSE

redis.LOG_NOTICE

redis.LOG_WARNING

它们与 Redis 的日志等级是对应的。
<h5 id="沙箱和最大执行时间">沙箱和最大执行时间</h5>
脚本不应该访问外部系统，包括文件系统和其他系统。脚本应该只能操作 Redis 数据和传入进来
参数。
脚本默认的最大执行时间是 5 秒（正常脚本执行时间都是毫秒级，所以 5 秒已经足够长了）。
以通过修改 lua-time-limit 变量来控制最大执行时间。
当脚本执行时间超过最大执行时间时，并不会被自动终止，因为这违反了脚本的原子性原则。当
个脚本执行时间过长时，Redis 会有如下操作：

Redis 记录下这个脚本执行时间过长

其他客户端开始接收命令，但是所有的命令都会会返回繁忙，除了 SCRIPT KILL 和 SHUTDOWN
NOSAVE

如果一个脚本仅执行只读命令，则可以用 SCRIPT KILL 命令来停止它。

如果脚本执行了写入命令，那么只能用 SHUTDOWN NOSAVE 来终止服务器，当前的所有数据不会保存到磁盘。

<h4 id="EVALSHA">EVALSHA</h4>
最早可用版本：2.6.0
用法：<code>EVALSHA sha1 numkeys key [key ...] arg [arg ...]</code>
该命令用来执行缓存在服务器上的脚本，sha1 为脚本的唯一标识。
使用 EVAL 命令必须每次都要把脚本从客户端传到服务器，由于 Redis 的内部缓存机制，它并不每次都重新编译脚本，但是传输上仍然浪费带宽。
另一方面，如果使用特殊命令或者通过 redis.conf 来定义命令会有以下问题：

不同实例有不同的实现方式

发布将会很困难，特别是分布式环境，因为要保证所有实例都包含给定的命令

读应用程序代码时，由于它调用了服务端命令，会不清楚代码的语义

为了避免这些问题，同时避免浪费带宽，Redis 实现了 EVALSHA 命令。
如果服务器中没有缓存指定的脚本，会返回给客户端脚本不存在的错误信息。
<h4 id="SCRIPT-DEBUG">SCRIPT DEBUG</h4>
最早可用版本：3.2.0
时间复杂度：O(1)
用法：<code>SCRIPT DEBUG YES|SYNC|NO</code>
该命令用于设置随后执行的 EVAL 命令的调试模式。Redis 包含一个完整的 Lua 调试器，代号为 DB，可以使编写复杂脚本的任务更加简单，在调试模式下，Redis 充当远程调试服务器，客户端可以步执行脚本，设置断点，检查变量等。想了解更多调试器内容的可以查看官方文档 Redis Lua debugger。
LDB 可以设置成异步或同步模式。异步模式下，服务器会 fork 出一个调试会话，不会阻塞主会，，调试会话结束后，所有数据都会回滚。同步模式则会阻塞会话，并保留调试过程中数据的改变。
<h4 id="SCRIPT-EXISTS">SCRIPT EXISTS</h4>
最早可用版本：2.6.0
时间复杂度：O(N)，N 是脚本数量
返回脚本是否存在于缓存中（存在返回 1，不存在返回 0）。这个命令适合在管道前执行，以保管道中的所有脚本都已经加载到服务器端了，如果没有，需要用 SCRIPT LOAD 命令进行加载。
<h4 id="SCRIPT-FLUSH">SCRIPT FLUSH</h4>
最早可用版本：2.6.0
时间复杂度：O(N)，N 是缓存中的脚本数
刷新缓存中的脚本，这一命令常在云服务上被使用。
<h4 id="SCRIPT-KILL">SCRIPT KILL</h4>
最早可用版本：2.6.0
时间复杂度：O(1)
停止当前正在执行的 Lua 脚本，通常用来停止执行时间过长的脚本。停止后，被阻塞的客户端抛出一个错误。
<h4 id="SCRIPT-LOAD">SCRIPT LOAD</h4>

最早可用版本：2.6.0
时间复杂度：O(N)，N 是脚本的字节数
该命令用于将脚本加载到服务器端的缓存中，但不会执行。加载后，服务器会一直缓存，因为良的应用程序不太可能有太多不同的脚本导致内存不足。每个脚本都像一个新命令的缓存，所以即使是型应用程序，也就有几百个，它们占用的内存是微不足道的。
<h4 id="小结">小结</h4>
本文介绍了 Redis Lua 相关的命令。其中 EVAL 和 EVALSHA 用来执行脚本。脚本执行具有原性。脚本的复制和传播可以根据需要设置。脚本中不能定义全局变量。
作者：Jackeyzhe
链接：https://www.jianshu.com/p/fe512264e0 0
来源：简书
