随笔 - 19  文章 - 0  评论 - 297

[**【菜鸟玩Linux开发】通过MySQL自动同步刷新Redis**](http://www.cnblogs.com/zhxilin/p/5923671.html)

在服务端开发过程中，一般会使用MySQL等关系型数据库作为最终的存储引擎，Redis其实也可以作为一种键值对型的数据库，但在一些实际场景中，特别是关系型结构并不适合使用Redis直接作为数据库。这俩家伙简直可以用“男女搭配，干活不累”来形容，搭配起来使用才能事半功倍。本篇我们就这两者如何合理搭配以及他们之间数据如何进行同步展开。

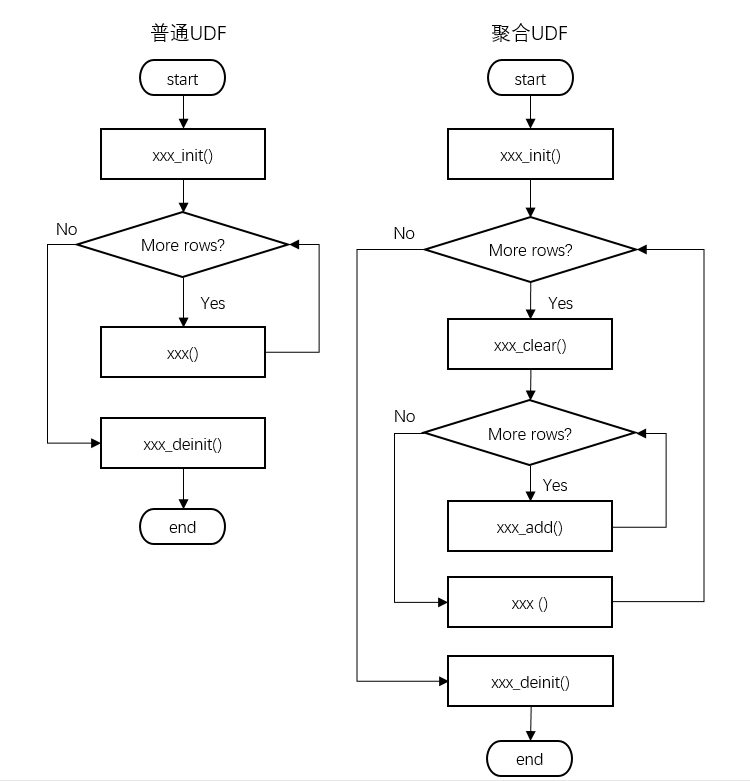
一般地，Redis可以用来作为MySQL的缓存层。为什么MySQL最好有缓存层呢？想象一下这样的场景：在一个多人在线的游戏里，排行榜、好友关系、队列等直接关系数据的情景下，如果直接和MySQL正面交手，大量的数据请求可能会让MySQL疲惫不堪，甚至过量的请求将会击穿数据库，导致整个数据服务中断，数据库性能的瓶颈将掣肘业务的开发；那么如果通过Redis来做数据缓存，将大大减小查询数据的压力。在这种架子里，当我们在业务层有数据查询需求时，先到Redis缓存中查询，如果查不到，再到MySQL数据库中查询，同时将查到的数据更新到Redis里；当我们在业务层有修改插入数据需求时，直接向MySQL发起请求，同时更新Redis缓存。

在上面这种架子中，有一个关键点，就是MySQL的CRUD发生后自动地更新到Redis里，这需要通过MySQL UDF来实现。具体来说，我们把更新Redis的逻辑放到MySQL中去做，即定义一个触发器Trigger，监听CRUD这些操作，当操作发生后，调用对应的UDF函数，远程写回Redis，所以业务逻辑只需要负责更新MySQL就行了，剩下的交给MySQL UDF去完成。

**一. 什么是UDF**

UDF，是User Defined Function的缩写，用户定义函数。MySQL支持函数，也支持自定义的函数。UDF比存储方法有更高的执行效率，并且支持聚集函数。

UDF定义了5个API：xxx\_init()、xxx\_deinit()、xxx()、xxx\_add()、xxx\_clear()。[官方文档（http://dev.mysql.com/doc/refman/5.7/en/adding-udf.html）](http://dev.mysql.com/doc/refman/5.7/en/adding-udf.html" \t "_blank)给出了这些API的说明。相关的结构体定义在mysql\_com.h里，它又被mysql.h包含，使用时只需#include<mysql.h>即可。他们之间的关系和执行顺序可以以下图来表示：



**1. xxx()**

这是主函数，5个函数至少需要xxx()，对MySQL操作的结果在此返回。函数的声明如下：

char \*xxx(UDF\_INIT \*initid, UDF\_ARGS \*args, char \*result, unsigned long \*length, char \*is\_null, char \*error);

long long xxx(UDF\_INIT \*initid, UDF\_ARGS \*args, char \*is\_null, char \*error);

double xxx(UDF\_INIT \*initid, UDF\_ARGS \*args, char \*is\_null, char \*error);

SQL的类型和C/C++类型的映射：

|  |  |
| --- | --- |
| **SQL Type** | **C/C++ Type** |
| STRING | char \* |
| INTEGER | long long |
| REAL | double |

**2. xxx\_init()**

xxx()主函数的初始化，如果定义了，则用来检查传入xxx()的参数数量、类型、分配内存空间等初始化操作。函数的声明如下：

my\_bool xxx\_init(UDF\_INIT \*initid, UDF\_ARGS \*args, char \*message);

**3. xxx\_deinit()**

xxx()主函数的反初始化，如果定义了，则用来释放初始化时分配的内存空间。函数的声明如下：

void xxx\_deinit(UDF\_INIT \*initid);

**4. xxx\_add()**

在聚合UDF中反复调用，将参数加入聚合参数中。函数的声明如下：

void xxx\_add(UDF\_INIT \*initid, UDF\_ARGS \*args, char \*is\_null,char \*error);

**5. xxx\_clear()**

在聚合UDF中反复调用，重置聚合参数，为下一行数据的操作做准备。函数的声明如下：

void xxx\_clear(UDF\_INIT \*initid, char \*is\_null, char \*error);

**二. UDF函数的基本使用**

在此之前，需要先安装mysql的开发包：

[root@localhost zhxilin]# yum install mysql-devel -y

我们定义一个最简单的UDF主函数：

[复制代码](javascript:void(0);)

1 /\*simple.cpp\*/

2 #include <mysql.h>

3

4 extern "C" long long simple\_add(UDF\_INIT \*initid, UDF\_ARGS \*args, char \*is\_null, char \*error)

5 {

6 int a = \*((long long \*)args->args[0]);

7 int b = \*((long long \*)args->args[1]);

8 return a + b;

9 }

10

11 extern "C" my\_bool simple\_add\_init(UDF\_INIT \*initid, UDF\_ARGS \*args, char \*message)

12 {

13 return 0;

14 }

[复制代码](javascript:void(0);)

由于mysql提供的接口是C实现的，我们在C++中使用时需要添加：

extern "C" { ... }

接下来编译成动态库.so：

[zhxilin@localhost mysql-redis-test]$ g++ **-shared -fPIC** -I /usr/include/mysql -o simple\_add.so simple.cpp

**-shared 表示编译和链接时使用的是全局共享的类库;**

**-fPIC编译器输出位置无关的目标代码，适用于动态库；**

-I /usr/include/mysql 指明包含的头文件mysql.h所在的位置。

编译出simple\_add.so后用root拷贝到/usr/lib64/mysql/plugin下：

[root@localhost mysql-redis-test]# cp simple\_add.so /usr/lib64/mysql/plugin/

紧接着可以在MySQL中创建函数执行了。登录MySQL，创建关联函数：

mysql> CREATE FUNCTION simple\_add RETURNS INTEGER SONAME 'simple\_add.so';

Query OK, **0** rows affected (**0.04** sec)

测试UDF函数：

[复制代码](javascript:void(0);)

mysql> select simple\_add(**10**, **5**);

+-------------------+

| simple\_add(**10**, **5**) |

+-------------------+

| **15** |

+-------------------+

**1** row in set (**0.00** sec)

[复制代码](javascript:void(0);)

可以看到，UDF正确执行了加法。

创建UDF函数的语法是 CREATE FUNCTION xxx RETURNS [INTEGER/STRING/REAL] SONAME '[so name]';

删除UDF函数的语法是 DROP FUNCTION simple\_add;

mysql> DROP FUNCTION simple\_add;

Query OK, **0** rows affected (**0.03** sec)

**三. 在UDF中访问Redis**

跟上述做法一样，只需在UDF里调用Redis提供的接口函数。Redis官方给出了[Redis C++ Client （https://github.com/mrpi/redis-cplusplus-client）](https://github.com/mrpi/redis-cplusplus-client" \t "_blank)，封装了Redis的基本操作。

源码是依赖boost，需要先安装boost：

[root@localhost dev]# yum install boost boost-devel

然后下载redis cpp client源码：

[root@localhost dev]# git clone https://github.com/mrpi/redis-cplusplus-client

使用时需要把redisclient.h、anet.h、fmacros.h、anet.c 这4个文件考到目录下，开始编写关于Redis的UDF。我们定义了redis\_hset作为主函数，连接Redis并调用hset插入哈希表，redis\_hset\_init作为初始化，检查参数个数和类型。

[复制代码](javascript:void(0);)

1 /\* test.cpp \*/

2 #include <stdio.h>

3 #include <mysql.h>

4 #include "redisclient.h"

5 using namespace boost;

6 using namespace std;

7

8 static redis::client \*m\_client = NULL;

9

10 extern "C" char \*redis\_hset(UDF\_INIT \*initid, UDF\_ARGS \*args, char \*result, unsigned long \*length, char \*is\_null, char \*error) {

11 try {

12 // 连接Redis

13 if(NULL == m\_client) {

14 const char\* c\_host = getenv("REDIS\_HOST");

15 string host = "127.0.0.1";

16 if(c\_host) {

17 host = c\_host;

18 }

19 m\_client = new redis::client(host);

20 }

21

22 if(!(args->args && args->args[0] && args->args[1] && args->args[2])) {

23 \*is\_null = 1;

24 return result;

25 }

26

27 // 调用hset插入一个哈希表

28 if(m\_client->hset(args->args[0], args->args[1], args->args[2])) {

29 return result;

30 } else {

31 \*error = 1;

32 return result;

33 }

34 } catch (const redis::redis\_error& e) {

35 return result;

36 }

37 }

38

39 extern "C" my\_bool redis\_hset\_init(UDF\_INIT \*initid, UDF\_ARGS \*args, char \*message) {

40 if (3 != args->arg\_count) {

41 // hset(key, field, value) 需要三个参数

42 strncpy(message, "Please input 3 args for: hset('key', 'field', 'value');", MYSQL\_ERRMSG\_SIZE);

43 return -1;

44 }

45 if (args->arg\_type[0] != STRING\_RESULT ||

46 args->arg\_type[1] != STRING\_RESULT ||

47 args->arg\_type[2] != STRING\_RESULT) {

48 // 检查参数类型

49 strncpy(message, "Args type error: hset('key', 'field', 'value');", MYSQL\_ERRMSG\_SIZE);

50 return -1;

51 }

52

53 args->arg\_type[0] = STRING\_RESULT;

54 args->arg\_type[1] = STRING\_RESULT;

55 args->arg\_type[2] = STRING\_RESULT;

56

57 initid->ptr = NULL;

58 return 0;

59 }

[复制代码](javascript:void(0);)

编译链接：

[zhxilin@localhost mysql-redis-test]$ g++ -shared -fPIC -I /usr/include/mysql -lboost\_serialization -lboost\_system -lboost\_thread -o libmyredis.so anet.c test.cpp

编译时需要加上-lboost\_serialization -lboost\_system -lboost\_thread， 表示需要链接三个动态库：libboost\_serialization.so、libboost\_system.so、libboost\_thread.so，否则在运行时会报缺少函数定义的错误。

编译出libmyredis.so之后，将其拷贝到mysql的插件目录下并提权：

[root@localhost mysql-redis-test]# cp libmyredis.so /usr/lib64/mysql/plugin/ & chmod 777 /usr/lib64/mysql/plugin/libmyredis.so

完成之后登录MySQL，创建关联函数测试一下：

mysql> DROP FUNCTION IF EXISTS `redis\_hset`;

Query OK, **0** rows affected (**0.16** sec)

mysql> CREATE FUNCTION redis\_hset RETURNS STRING SONAME 'libmyredis.so';

Query OK, **0** rows affected (**0.02** sec)

先删除老的UDF，注意函数名加反引号（``）。调用UDF测试，返回0，执行成功：

[复制代码](javascript:void(0);)

mysql> SELECT redis\_hset('zhxilin', 'id', '09388334');

+-----------------------------------------+

| redis\_hset('zhxilin', 'id', '09388334') |

+-----------------------------------------+

| **0** |

+-----------------------------------------+

**1** row in set (**0.00** sec)

[复制代码](javascript:void(0);)

打开redis-cli，查看结果：

**127.0**.**0.1**:**6379**> HGETALL zhxilin

**1**) "id"

**2**) "**09388334**"

**四. 通过MySQL触发器刷新Redis**

 在上一节的基础上，我们想让MySQL在增删改查的时候自动调用UDF，还需要借助MySQL触发器。触发器可以监听INSERT、UPDATE、DELETE等基本操作。在MySQL中，创建触发器的基本语法如下：

CREATE TRIGGER trigger\_name

trigger\_time

trigger\_event ON table\_name

FOR EACH ROW

trigger\_statement

trigger\_time表示触发时机，值为**AFTER**或**BEFORE**；

trigger\_event表示触发的事件，值为**INSERT**、**UPDATE**、**DELETE**等；

trigger\_statement表示触发器的程序体，可以是一句SQL语句或者调用UDF。

在trigger\_statement中，如果有多条SQL语句，需要用BEGIN...END包含起来：

BEGIN

[statement\_list]

END

由于MySQL默认的结束分隔符是分号（;），如果我们在BEGIN...END中出现了分号，将被标记成结束，此时没法完成触发器的定义。有一个办法，可以调用DELIMITER命令来暂时修改结束分隔符，用完再改会分号即可。比如改成$：

mysql> DELIMITER $

我们开始定义一个触发器，监听对Student表的插入操作，Student表在上一篇文章中创建的，可以查看[上一篇](http://www.cnblogs.com/zhxilin/p/5897211.html" \t "_blank)文章。

[复制代码](javascript:void(0);)

mysql > DELIMITER $

> CREATE TRIGGER tg\_student

> AFTER INSERT on Student

> FOR EACH ROW

> BEGIN

> SET @id = (SELECT redis\_hset(CONCAT('stu\_', new.Sid), 'id', CAST(new.Sid AS CHAR(**8**))));

> SET @name = (SELECT redis\_hset(CONCAT('stu\_', new.Sid), 'name', CAST(new.Sname AS CHAR(**20**))));

> Set @age = (SELECT redis\_hset(CONCAT('stu\_', new.Sid), 'age', CAST(new.Sage AS CHAR)));

> Set @gender = (SELECT redis\_hset(CONCAT('stu\_', new.Sid), 'gender', CAST(new.Sgen AS CHAR)));

> Set @dept = (SELECT redis\_hset(CONCAT('stu\_', new.Sid), 'department', CAST(new.Sdept AS CHAR(**10**))));

> END $

[复制代码](javascript:void(0);)

创建完触发器可以通过show查看，或者drop删除：

mysql> SHOW TRIGGERS;

mysql> DROP TRIGGER tg\_student;

接下来我们调用一句插入语句，然后观察Redis和MySQL数据的变化：

mysql> INSERT INTO Student VALUES('09388165', 'Rose', **19**, 'F', 'SS3-205');

Query OK, **1** row affected (**0.27** sec)

MySQL的结果：

[复制代码](javascript:void(0);)

mysql> SELECT \* FROM Student;

+----------+---------+------+------+---------+

| Sid | Sname | Sage | Sgen | Sdept |

+----------+---------+------+------+---------+

| **09388123** | Lucy | **18** | F | AS2-**123** |

| **09388165** | Rose | **19** | F | SS3-**205** |

| **09388308** | zhsuiy | **19** | F | MD8-**208** |

| **09388318** | daemon | **18** | M | ZS4-**630** |

| **09388321** | David | **20** | M | ZS4-**731** |

| **09388334** | zhxilin | **20** | M | ZS4-**722** |

+----------+---------+------+------+---------+

**6** rows in set (**0.00** sec)

[复制代码](javascript:void(0);)

Redis的结果：

[复制代码](javascript:void(0);)

**127.0**.**0.1**:**6379**> HGETALL stu\_09388165

**1**) "id"

**2**) "**09388165**"

**3**) "name"

**4**) "Rose"

**5**) "age"

**6**) "**19**"

**7**) "gender"

**8**) "F"

**9**) "department"

**10**) "SS3-**205**"

[复制代码](javascript:void(0);)

以上结果表明，当MySQL插入数据时，通过触发器调用UDF，实现了自动刷新Redis的数据。另外，调用MySQL插入的命令，可以通过C++实现，进而就实现了在C++的业务逻辑里，只需调用MySQL++的接口就能实现MySQL数据库和Redis缓存的更新，这部分内容在[上一篇文章](http://www.cnblogs.com/zhxilin/p/5897211.html" \t "_blank)已经介绍过了。

**总结**

通过实践，能体会到MySQL和Redis是多么相亲相爱吧！^\_^

本篇文章讲了从最基础的UDF开始，再到通过UDF连接Redis插入数据，再进一步介绍通过MySQL Trigger自动更新Redis数据的整个思路，实现了一个目标，即只在业务代码中更新MySQL数据库，进而Redis能够自动同步刷新。

MySQL对UDF函数和触发器的支持，使得实现Redis数据和MySQL自动同步成了可能。当然UDF毕竟是通过插件的形式运行在MySQL中的，并没有过多的安全干预，一旦插件发生致命性崩溃，有可能MySQL也会挂，所以在编写UDF的时候需要非常谨慎！