

云计算技术方案

1、项目需求分析

基于云服务器环境和数据同步需求，我们需要设计一套完整的银行业务交易系统的数据同步解决方案并给予实现。针对于有较高联机访问量的某银行金融账务交易系统，该方案需要实现以下需求：

- 1) 搭建数据库，设计数据表，将数据导入；
- 2) 搭建模拟的金融业务应用，模拟客户维护（姓名、地址），账户存款、账户取款等 3 个金融业务场景连续不间断发生；要求该应用在短时间内，完成随机 20 万张客户和 20 万张账户的上述业务；
- 3) 准实时同步，设计独立的数据信息同步的处理机制，要求在以上模拟应用程序运行导致数据变化的同时，极短时延内快速准确将每条变化的记录同步到备库；
- 4) 全量同步，假定主库绝大部分数据表需要同步，需设计一套数据库主库数据同步方案，需于每日日初固定时点，在尽可能短的时延内，自动将主库所有数据同步至备库。

2、总体设计方案

2.1 数据库表的设计

通过分析项目需求，我们需要在 MySQL 中建立四个表来存储所提供的客户信息 1、客户信息 2、账户流水及客户账户四份原始数据。我们通过：

```
create table accountfile(  
    account CHARACTER(12),  
    status CHARACTER(12),  
    opendate DATE,  
    lastexchange DATE,  
    balance FLOAT(17,3),  
    uid CHARACTER(7),  
    salary FLOAT(17,3),  
    PRIMARY KEY (account),
```

```
FOREIGN KEY(uid) REFERENCES userinfo1(uid)
)ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

```
create table accountliquidfile (
    account CHARACTER(12),
    time INTEGER,
    iquidnum CHARACTER(9),
    date DATE,
    exchange FLOAT(17,3),
    balance FLOAT(17,3)
)ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

```
create table userinfo1 (
    uid CHARACTER(7),
    name CHARACTER(20),
    address CHARACTER(80),
    post CHARACTER(10),
    phone CHARACTER(16),
    PRIMARY KEY (uid)
)ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

```
create table userinfo2 (
    uid CHARACTER(7),
    idcard CHARACTER(22),
    birthday DATE,
    sex CHARACTER(1),
    marry CHARACTER(1),
    country CHARACTER(2),
    job CHARACTER(2),
    PRIMARY KEY (uid)
)ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

四条语句分别创建 accountfile、accountliquidfile、userinfo1、userinfo2 四个

表来存储对应信息。

2.1 数据库表的写入

在创建好数据库表后，我们需要将原始 txt 文本数据分别写入数据库表中。在读取文本数据时，我们采用按行解析、按空格分割的方法来读取数据。具体操作流程如下：

1) 连接数据库；

```
//数据库连接
void connect_database(MYSQL *mysql){
    //定义句柄
    //初始化句柄
    if(NULL == mysql_init(mysql))
    {
        printf("mysql init error!\n");
        exit(1);
        //return -1;
    }

    if(NULL == mysql_real_connect(mysql,"localhost","root","",,"bank",0,NULL,0))
    {
        printf("%s\n", mysql_error(mysql));
        exit(1);
        //return -1;
    }
    printf("连接数据库成功! \n");
    //设置字符集
    mysql_set_character_set(mysql,"utf8");
}
```

图 1 连接数据库

2) 按行解析，按空格分隔数据，并写入对应表中；

```
//解析一行数据，按空格分割，返回值放入res
void parse_line(char * input,char *res[]){

    char *p = strtok(input, " ");

    int i=0;
    if(p){
        res[i++]=p;
    }

    while(p=strtok(NULL, " ")){//使用第一个参数为NULL来提取子串
        res[i++]=p;
    }
}
```

图 2 按行解析数据

我们按照上述步骤操作后，就可以得到插入了原始数据的 `accountfile`、`accountliquidfile`、`userinfo1`、`userinfo2` 四个表。四张表之间的设计和连接关系如下所示：

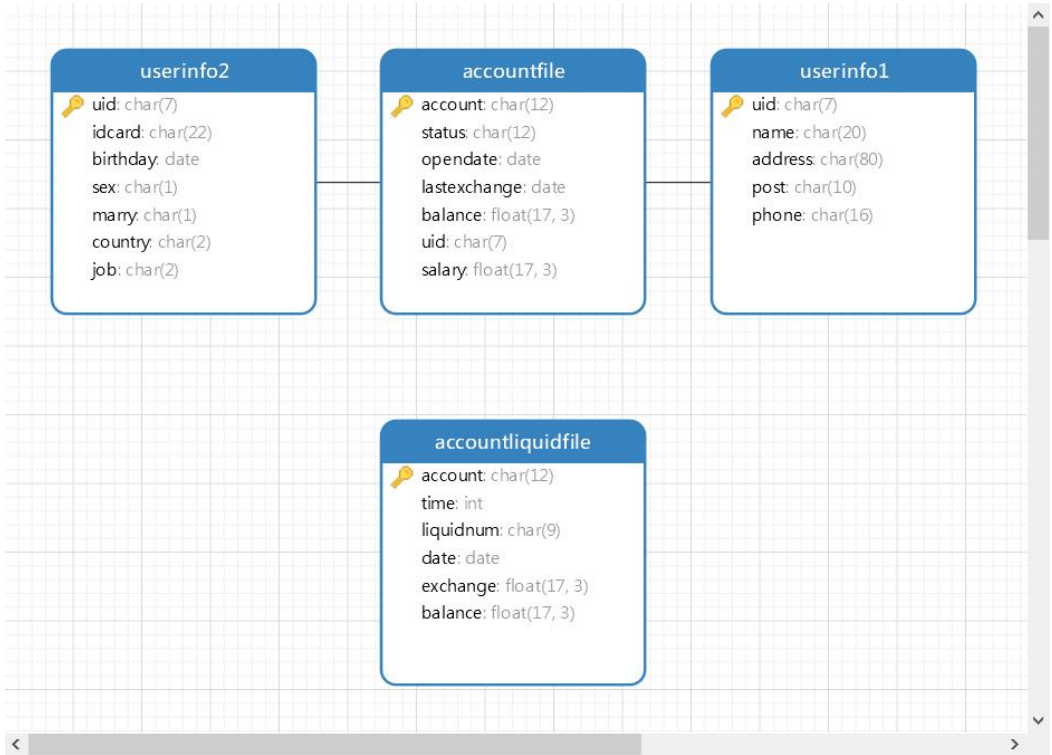


图 6 数据库 E-R 图

2.2 数据库表的增删改查

2.2.1 客户端与服务器通信接口说明

1) 帐户入金

英文名称	中文名称	数据类型	数据长度	是否必输	备注
输入					
TranCd	交易代码	String	8	Y	GDRC_001
AcctNo	客户账号	String	12	Y	
TranDate	交易日期	String	8	N	自然日
TranAmt	交易金额	Double	17, 3	Y	正数为入客户账，负数为扣客户账（余额不能为负）

输出					
TranCd	交易代码	String	8	Y	交易代码
AcctNo	客户账号	String	12	Y	
CustNo	客户号	String	7	Y	
CurrBal	当前余额	Double	17, 3	Y	
RetCd	交易返回 代码	String	6	Y	成功返回 “000000”； 错误返回 “000001”
RetMsg	交易返回 信息	String	255	Y	后台系统对业 务成功/错误 的描述信息

2) 更新客户电话号码

英文名称	中文名称	数据类型	数据长度	是否必输	备注
输入					
TranCd	交易代码	String	8	Y	GDRC_002
AcctNo	客户账号	String	12	Y	
TranDate	交易日期	String	8	N	自然日
PhoneNo	电话号码	String	15	Y	
输出					
TranCd	交易代码	String	8	Y	交易代码
AcctNo	客户账号	String	12	Y	
CustNo	客户号	String	7	Y	
PhoneNo	电话号码	String	15	Y	当前电话号码
RetCd	交易返回代 码	String	6	Y	成功返回 “000000”； 错误返回 “000001”

RetMsg	交易返回信息	String	255	Y	后台系统对业务成功/错误的描述信息
--------	--------	--------	-----	---	-------------------

3) 查询余额

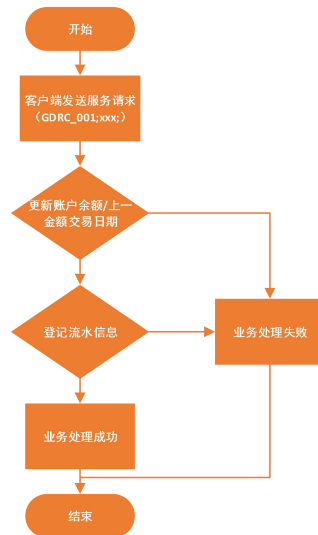
英文名称	中文名称	数据类型	数据长度	是否必输	备注
输入					
TranCd	交易代码	String	8	Y	GDRC_003
AcctNo	客户账号	String	12	Y	
输出					
TranCd	交易代码	String	8	Y	交易代码
AcctNo	客户账号	String	12	Y	
CustNo	客户号	String	7	Y	
Balance	账户余额	String	15	Y	当前帐户余额
RetCd	交易返回代码	String	6	Y	成功返回“000000”; 错误返回“000001”
RetMsg	交易返回信息	String	255	Y	后台系统对业务成功/错误的描述信息

2.2.2 具体业务说明

1) 查询余额

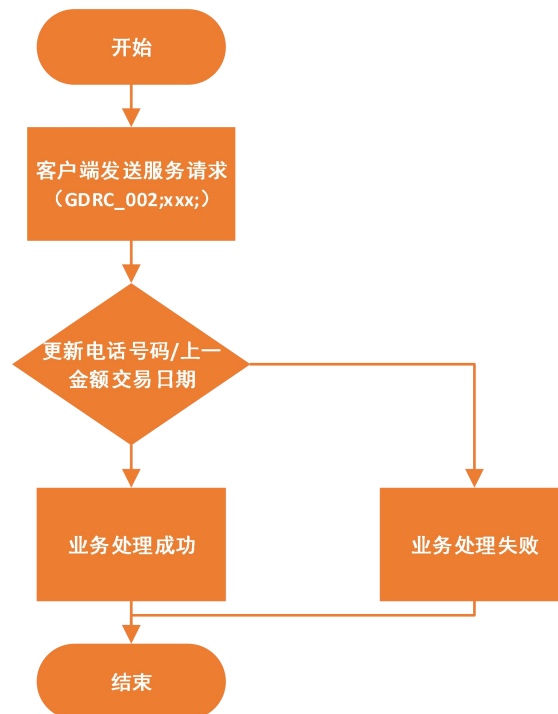
查询余额客户端通过向服务器发送 GDRC_003 指令以及用户所要查询某个账户的账户号对 Redis 缓存进行查询，通过查询 accountfile 表中与客户端发送的 uid 和账户号与表中相对应的词条记录中的 balance 值，如下图所示。

存钱操作客户端通过向服务器发送操作码 GDRC_001 以及用户 uid 以及存钱金额、用户所要存钱的帐户号、流水号。因为此操作要改变数据库中的值，因此存钱操作通过 "UPDATE accountfile SET accountfile.balance = accountfile.balance + " + amount + " WHERE uid = " + uid + " AND account = " + account + ";" 改变数据库中的帐户余额值，且需要在 accountliquidfile 表中增添一条流水数据信息通过 "INSERT INTO accountliquidfile (account, time, liquidnum, date, exchange, balance) VALUES(" + account + ", " + time + ", " + liquidnum + ", " + date + ", " + amount + ", " + balance[0] + ");"；更改完成之后需要更新 Redis 缓存的值使之与数据库中的值完成同步，如下图所示。



3) 修改个人信息

修改个人信息操作客户端通过向服务器发送操作码 GDRC_002 以及用户 uid 以及用户需要修改的地址，邮编，手机号码，结婚，国家，工作信息。因为此操作要改变数据库中的值，因此修改个人信息操作通过"UPDATE userinfo1 u1, userinfo2 u2 SET u1.address = '' + address + '', u1.post = '' + post + '', u1.phone = '' + phone + '', u2.marry = '' + marry + '', u2.country = '' + country + '', u2.job = '' + job + '' WHERE u1.uid = u2.uid AND u1.uid = '' + uid + ';"更改数据库中的用户个人信息;更改完成之后需要更新 Redis 缓存的值使之与数据库中的值完成同步。



2.2.3 数据库增删改查测试

对用户 uid 为 1573716 的用户账户号为 000002274525 的帐户余额进行查询，查询到的余额结果为 6555.250 元，如图 7 所示。

```
(base) → curd ./client
请操作：执行完毕后ctrl+c退出！
11;1573761;000002274525;
n=[8],buf=[6555.250]
```

图 7 余额查询

对用户 uid 为 1573716 的用户账户号为 000002274525 的帐户进行存钱，存款金额为 1000 元，流水帐号为 123456789。服务端若存钱成功则给客户端返回一个“yes”字符串。

```
31;1573761;1000;000002274525;123456789;
n=[3],buf=[yes]
```

图 8 存钱操作

对帐户存钱之后再次查询余额，则查询到的结果为存钱之前的结果加上存钱的金额，此处为 7555.250，如图 9 所示。

```
11;1573761;000002274525;
n=[8],buf=[7555.250]
```

图 9 查询余额

对用户 uid 为 1573716 的用户账户号为 000002274525 的帐户进行取钱操作，取钱金额为 1000 元，流水帐号为 123456789。服务端若操作成功则给客户端返回一个“yes”字符串。

```
32;1573761;1000;000002274525;123456789;
n=[3],buf=[yes]
```

图 10 取钱操作

对帐户取钱之后再次查询余额，则查询到的结果为取钱之前的结果加上本次取钱的金额，本次操作的结果为 6555.250，如图 11 所示。

```
11;1573761;000002274525;
n=[8],buf=[6555.250]
```

图 11 查询余额

2.3 epoll 模型处理高并发

2.3.1 高并发模型的选择

一般的高并发的服务器端的设计技术包括：多进程；多线程；select 模型；poll 模型；epoll 模型。我们这里使用的是效率高的 epoll 模型来处理高并发的场景。

首先，考虑到多进程和多线程的服务器对于在处理高并发的场景时，会占用大量的系统资源，尤其是在客户端连接上服务器之后，却长时间不进行通信，这样会占用一个子线程或者子进程用来维护这个客户端的通信，使得本来需要与服务器端通信的客户迟迟得不到响应。

其次，`select` 模型和 `epoll` 模型虽然也是委托内核监控文件描述符的变化，但是由于底层的数据结构的桎梏，导致内核每次只能返回发生变化的文件描述符的个数，却不能返回具体哪个文件描述符发生了变化。此时，就需要遍历所有的文件描述符，找到变化的文件描述符，然后进行通信。效率比较低下，尤其是在连接了大量客户端时，每次只有少量的客户端进行通信，这时花费了大量时间用来定位发生变化的文件描述符，大大降低了效率。

最后，我们选择 `epoll` 模型来处理高并发的客户端连接，因为 `epoll` 模型底层实现是红黑树，树上的每个节点都至少包括：文件描述符，事件类型，回调函数。每次会将发生变化的文件描述符对应的节点映射到一个链表返回给主函数，此时就不需要遍历所有的文件描述符，只需要处理链表中的节点即可，根据其事件类型，执行其对应的回调函数，完成与服务器端通信的功能。

2.3.2 基于 `epoll` 模型的服务器开发流程

1) 服务器端开发流程：

- (1) 创建 `socket`，得到监听文件描述符 `lfd`----`socket()`。
- (2) 设置端口复用----`setsockopt()`。
- (3) 绑定 `ip` 和端口----`bind()`。
- (4) 监听端口----`listen()`。
- (5) 创建一棵 `epoll` 树。
- (6) 将监听文件描述符 `lfd` 上 `epoll` 树。
- (7) 循环等待事件的发生。若是新的客户端到来，则主线程接受连接，并且将通信文件描述符 `cfid` 上树；若是客户端有数据发来，则利用通信文件描述符 `cfid` 进行通信。

2) 客户端开发流程：

- (1) 创建 `socket`，得到通信文件描述符 `sockfd`。
- (2) 向服务器端发起连接请求 `connect()`。

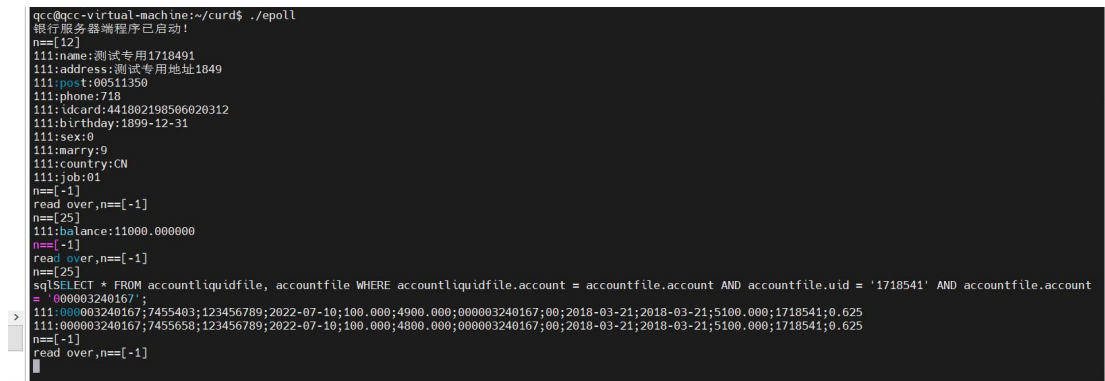
(3) 经过三次握手建立连接后，循环收发数据。

2.3.3 服务器程序与多客户端程序通信

以下总共开了四个终端，一个终端运行服务器端程序，三个终端运行客户端程序，分别执行不同的操作，与服务器端进行通信。

- (1) 图 12 运行的是服务器端程序；
- (2) 图 13 是第一个客户端程序，查询的是 uid 为 1718491 的客户的个人信息；
- (3) 图 14 是第二个客户端程序，查询的是 uid 为 1718491，账户号为 000003240086 的客户的账户余额；
- (4) 图 15 是第三个客户端程序，查询的是 uid 为 1718541，账户号为 000003240167 的客户的账户流水信息。

从运行结果可以看出，我们的服务器端程序支持与多客户端同时进行通信，而不互相影响，支持并发的处理客户端的请求。

A terminal window showing the execution of a server program. The prompt is 'qcc@qcc-virtual-machine:~/curd\$./epoll'. The program outputs '银行服务器端程序已启动!' and then enters a loop. It receives a connection from '111:测试专用1718491' and prints various user details like name, address, phone, etc. Then it receives another connection from '111:balance:11000.000000' and prints the balance. Finally, it receives a connection from '111:000003240167;7455403;123456789;2022-07-10;100.000;4900.000;000003240167;00;2018-03-21;2018-03-21;5100.000;1718541;0.625' and prints a long SQL query and its result. The terminal output is as follows:

```
qcc@qcc-virtual-machine:~/curd$ ./epoll
银行服务器端程序已启动!
n==[12]
111:name:测试专用1718491
111:address:测试专用地址1849
111:post:00511350
111:phone:718
111:icard:441802198506020312
111:birthday:1899-12-31
111:sex:0
111:marry:9
111:country:CN
111:job:01
n==[-1]
read over,n==[-1]
n==[25]
111:balance:11000.000000
n==[-1]
read over,n==[-1]
n==[25]
sqlSELECT * FROM accountliquidfile, accountfile WHERE accountliquidfile.account = accountfile.account AND accountfile.uid = '1718541' AND accountfile.account = '000003240167';
111:000003240167;7455403;123456789;2022-07-10;100.000;4900.000;000003240167;00;2018-03-21;2018-03-21;5100.000;1718541;0.625
111:000003240167;7455403;123456789;2022-07-10;100.000;4800.000;000003240167;00;2018-03-21;2018-03-21;5100.000;1718541;0.625
n==[-1]
read over,n==[-1]
```

图 12 服务器端程序

```
(SSH client, X server and network tools)

> SSH session to qcc@192.168.198.143
? Direct SSH : ✓
? SSH compression : ✓
? SSH-browser : ✓
? X11-forwarding : ✓ (remote display is forwarded through SSH)
> For more info, ctrl+click on help or visit our website.

Welcome to Ubuntu 18.04.6 LTS (GNU/Linux 5.4.0-121-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

37 updates can be applied immediately.
To see these additional updates run: apt list --upgradable

New release '20.04.4 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Your Hardware Enablement Stack (HWE) is supported until April 2023.
Last login: Mon Jul 11 15:23:30 2022 from 192.168.198.1
qcc@qcc-virtual-machine:~$ cd curd/
qcc@qcc-virtual-machine:~/curd$ ./client
请操作: 执行完后后ctrl+c退出!
11:1718491;
n=[0],buf=[]
^C
qcc@qcc-virtual-machine:~/curd$ ./client
请操作: 执行完后后ctrl+c退出!
13:1718491;
n=[126],buf=[name:测试专用1718491
address:测试专用地址1849
post:00511350
phone:718
idcard:441882198506020312
birthday:1899-12-31
]
```

图 13 查询个人信息

```
? MobaXterm Personal Edition v21.2 ?
(SSH client, X server and network tools)

> SSH session to qcc@192.168.198.143
? Direct SSH : ✓
? SSH compression : ✓
? SSH-browser : ✓
? X11-forwarding : ✓ (remote display is forwarded through SSH)
> For more info, ctrl+click on help or visit our website.

Welcome to Ubuntu 18.04.6 LTS (GNU/Linux 5.4.0-121-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

37 updates can be applied immediately.
To see these additional updates run: apt list --upgradable

New release '20.04.4 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Your Hardware Enablement Stack (HWE) is supported until April 2023.
Last login: Mon Jul 11 16:14:18 2022 from 192.168.198.1
qcc@qcc-virtual-machine:~$ cd curd/
qcc@qcc-virtual-machine:~/curd$ ./client
请操作: 执行完后后ctrl+c退出!
11:1718491;000003240086;
n=[0],buf=[]
^C
qcc@qcc-virtual-machine:~/curd$ ./client
请操作: 执行完后后ctrl+c退出!
11:1718491;000003240086;
n=[21],buf=[balance:11000.000000
]
```

图 14 查询账户余额

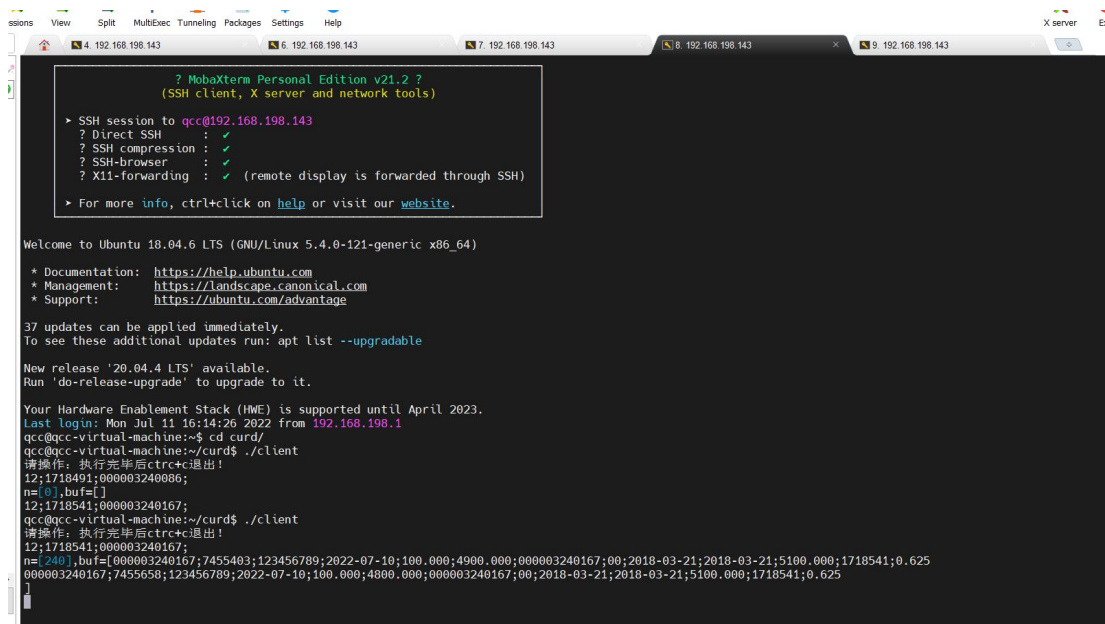


图 15 查询账户流水信息

2.4 Redis 缓存技术

2.4.1 Redis 缓存介绍

Redis 是一个开源（BSD 许可）的，内存中的数据结构存储系统，它可以用作数据库、缓存和消息中间件。它是基于高性能的 Key-Value、并提供多种语言的 API 的非关系型数据库。不过与传统数据库不同的是 Redis 的数据是存在内存中的，所以存写速度非常快。它支持多种类型的数据结构，如 字符串（strings），散列（hashes），列表（lists），集合（sets），有序集合（sorted sets）。

我们主要使用的数据结构是散列（hashes），通过键值对的形式，将数据库中的数据写入 Redis 缓存中，提高查询的效率。

2.4.2 sql 的设计与查询

1) 用户信息 1 表的 MySQL 到 Redis 的设计

从图 16 可以看出，userinfo1 表的主键是 uid。所以在设计 Redis 的哈希存储时，我们将（表名：主键）设置为 key，属性 name,address,post,phone 分别设置为不同的 field，其对应的值就是 value，在查询具体的某一条信息时，例如：通过 hget userinfo1:xxxxxxx name 命令就可以得到 userinfo1 表中 uid 为 xxxxxx 的客户的姓名。其他属性值查询也类似。

图 18 展示了 Redis 中存储的 userinfo1 的结果，每一行代表一个 key 值；

图 19 展示了在 Redis 中查询 userinfo1 表中 uid 为 1620028 的客户的姓名，

地址，邮编，电话各个字段的信息的结果。

```
mysql> desc userinfo1;
```

Field	Type	Null	Key	Default	Extra
uid	char(7)	NO	PRI	NULL	
name	char(20)	YES		NULL	
address	char(80)	YES		NULL	
post	char(10)	YES		NULL	
phone	char(16)	YES		NULL	

5 rows in set (0.11 sec)

图 16 数据库 userinfo1 表

```
select CONCAT('*10\r\n','$',LENGTH(redis_cmd),'\r\n',redis_cmd,'\r\n',
'$',LENGTH(redis_key),'\r\n',redis_key,'\r\n',
'$',LENGTH(hkey1),'\r\n',hkey1,'\r\n',
'$',LENGTH(hvalue1),'\r\n',hvalue1,'\r\n',
'$',LENGTH(hkey2),'\r\n',hkey2,'\r\n',
'$',LENGTH(hvalue2),'\r\n',hvalue2,'\r\n',
'$',LENGTH(hkey3),'\r\n',hkey3,'\r\n',
'$',LENGTH(hvalue3),'\r\n',hvalue3,'\r\n',
'$',LENGTH(hkey4),'\r\n',hkey4,'\r\n',
'$',LENGTH(hvalue4),'\r\n',hvalue4,'\r\n')
from (
select 'hset' as redis_cmd,
CONCAT('userinfo1:',uid) as redis_key,
'name' as hkey1,name as hvalue1,
'address' as hkey2,address as hvalue2,
'post' as hkey3,post as hvalue3,
'phone' as hkey4,phone as hvalue4
from userinfo1
) as a
```

图 17 用户信息 1 表的 sql 语句

```
qcc@qcc-virtual-machine:~/curd$ cd
qcc@qcc-virtual-machine:~$ redis-cli --raw
127.0.0.1:6379> auth qcc216
OK
127.0.0.1:6379> keys userinfo1
127.0.0.1:6379> keys userinfo1*
userinfo1:1620028
userinfo1:7367092
userinfo1:1614434
userinfo1:1794856
userinfo1:0203217
userinfo1:0939941
userinfo1:1632105
userinfo1:1697649
userinfo1:1821995
userinfo1:1641715
userinfo1:1532269
userinfo1:1521504
```

图 18 Redis 存储 userinfo1 表

```

127.0.0.1:6379> hget userinfo1:1620028 name
农信测试1620028
127.0.0.1:6379> hget userinfo1:1620028 address
测试专用地址2002
127.0.0.1:6379> hget userinfo1:1620028 post
00000000
127.0.0.1:6379> hget userinfo1:1620028 phone
620683821
127.0.0.1:6379>

```

图 19 Redis 查询 userinfo1 表

2) 用户信息 2 表的 MySQL 到 Redis 的设计

从图 20 可以看出，userinfo2 表的主键是 uid。所以在设计 Redis 的哈希存储时，我们将（表名：主键）设置为 key，属性 idcard、birthday、sex、marry、country、job 分别设置为不同的 field，其对应的值就是 value，在查询具体的某一条信息时，例如：通过 `hget userinfo2:xxxxxx idcard` 命令就可以得到 userinfo2 表中 uid 为 xxxxxx 的客户的身份证号。其他属性值查询也类似。

图 22 展示了 Redis 中存储的 userinfo2 的结果，每一行代表一个 key 值；

图 23 展示了在 Redis 中查询 userinfo2 表中 uid 为 065127 的客户的生日、性别、婚姻状态等各个字段的信息的结果。

```

mysql> desc userinfo2;
+-----+-----+-----+-----+-----+-----+
| Field | Type  | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| uid   | char(7) | NO   | PRI | NULL    |       |
| idcard | char(22) | YES  |     | NULL    |       |
| birthday | date   | YES  |     | NULL    |       |
| sex   | char(1) | YES  |     | NULL    |       |
| marry | char(1) | YES  |     | NULL    |       |
| country | char(2) | YES  |     | NULL    |       |
| job   | char(2) | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
7 rows in set (0.00 sec)

```

图 20 数据库中 userinfo2 表


```

select CONCAT('*14\r\n','$',LENGTH(redis_cmd),'\r\n',redis_cmd,'\r\n',
'$',LENGTH(redis_key),'\r\n',redis_key,'\r\n',
'$',LENGTH(hkey1),'\r\n',hkey1,'\r\n',
'$',LENGTH(hvalue1),'\r\n',hvalue1,'\r\n',
'$',LENGTH(hkey2),'\r\n',hkey2,'\r\n',
'$',LENGTH(hvalue2),'\r\n',hvalue2,'\r\n',
'$',LENGTH(hkey3),'\r\n',hkey3,'\r\n',
'$',LENGTH(hvalue3),'\r\n',hvalue3,'\r\n',
'$',LENGTH(hkey4),'\r\n',hkey4,'\r\n',
'$',LENGTH(hvalue4),'\r\n',hvalue4,'\r\n',
'$',LENGTH(hkey5),'\r\n',hkey5,'\r\n',
'$',LENGTH(hvalue5),'\r\n',hvalue5,'\r\n',
'$',LENGTH(hkey6),'\r\n',hkey6,'\r\n',
'$',LENGTH(hvalue6),'\r\n',hvalue6,'\r\n')
from (
select 'hset' as redis_cmd,
CONCAT('userinfo2:',uid) as redis_key,
'idcard' as hkey1,idcard as hvalue1,
'birthday' as hkey2,birthday as hvalue2,
'sex' as hkey3,sex as hvalue3,
'marry' as hkey4,marry as hvalue4,
'country' as hkey5,country as hvalue5,
'job' as hkey6,job as hvalue6
from userinfo2
) as b

```

图 21 用户 userinfo2 的 sql 语句

```

userinfo2:0187800
userinfo2:0992018
userinfo2:0659786
userinfo2:0292279
userinfo2:0094674
userinfo2:1480983
userinfo2:0911771
userinfo2:1537462
userinfo2:1499000
userinfo2:1097641
userinfo2:1243024
userinfo2:0841487
userinfo2:1023539
userinfo2:1162148
userinfo2:0999811
userinfo2:1514617
userinfo2:0314041
userinfo2:0927674
userinfo2:1170348
userinfo2:1321633
userinfo2:0651627
127.0.0.1:6379>

```

图 22 Redis 存储 userinfo2 表

```

OK
127.0.0.1:6379> hget userinfo2:0651627 idcard
442829530927493
127.0.0.1:6379> hget userinfo2:0651627 birthday
1999-12-31
127.0.0.1:6379> hget userinfo2:0651627 sex
1
127.0.0.1:6379> hget userinfo2:0651627 marry
7
127.0.0.1:6379> hget userinfo2:0651627 country
CN
127.0.0.1:6379> hget userinfo2:0651627 job
23
127.0.0.1:6379>

```

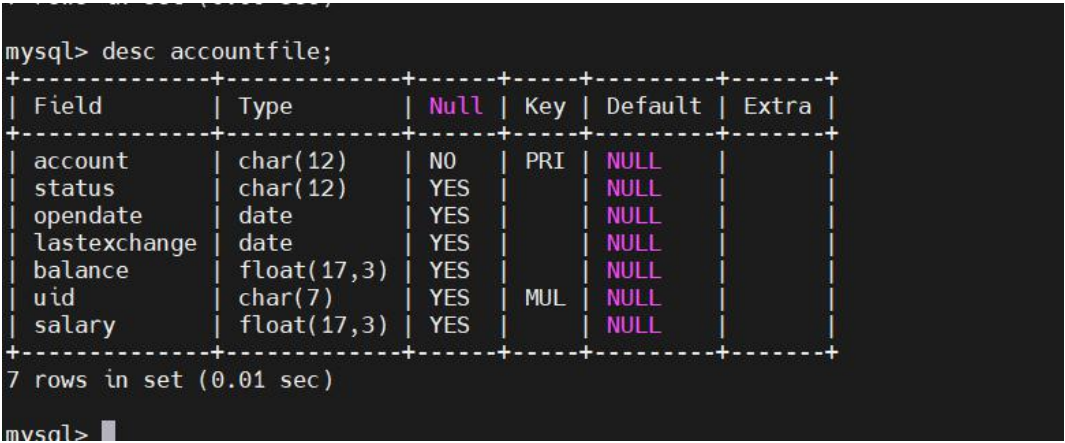
图 23 Redis 查询 userinfo2 表

3) 账户信息表的 MySQL 到 Redis 的设计

从图 24 可以看出，accountfile 表的主键是 account。外键是 uid。所以在设计 Redis 的哈希存储时，我们将（表名：主键）设置为 key，属性 status、opendata、lastexchange、balance、salary 分别设置为不同的 field，其对应的值就是 value，在查询具体的某一条信息时，例如：通过 hget accountfile:xxxxxx opendata 命令就可以得到 accountfile 表中 account 为 xxxxxx 的客户的开户时间。其他属性值查询也类似。

图 26 展示了 Redis 中存储的 accountfile 的结果，每一行代表一个 key 值；

图 27 展示了在 Redis 中查询 accountfile 表中 account 为 000000069079 的客户的开户日期、利息、上次交易时间等各个字段的信息的结果。



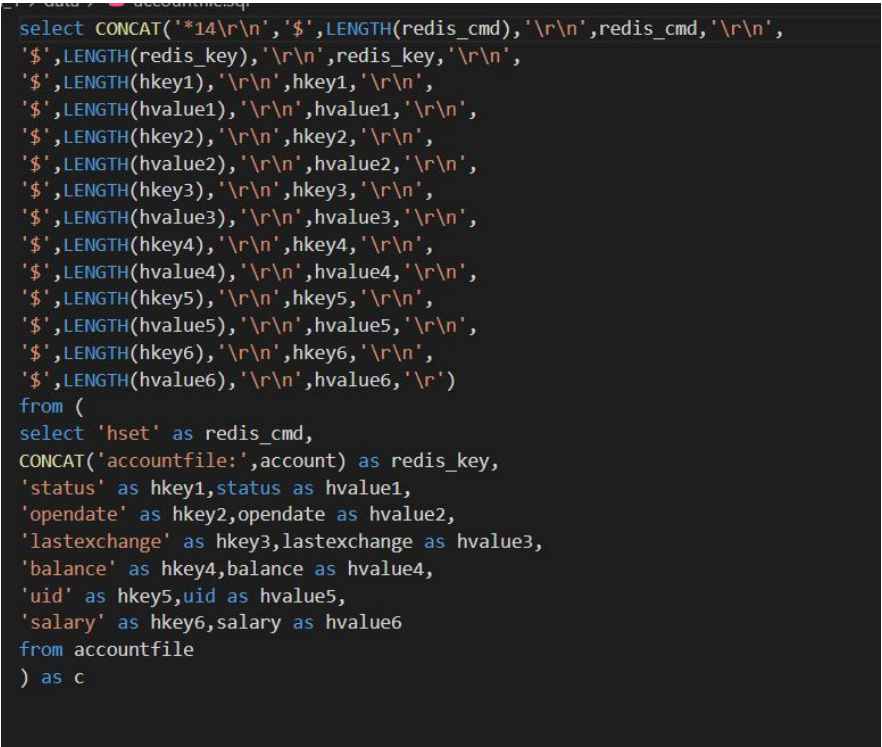
```
mysql> desc accountfile;
```

Field	Type	Null	Key	Default	Extra
account	char(12)	NO	PRI	NULL	
status	char(12)	YES		NULL	
opendate	date	YES		NULL	
lastexchange	date	YES		NULL	
balance	float(17,3)	YES		NULL	
uid	char(7)	YES	MUL	NULL	
salary	float(17,3)	YES		NULL	

7 rows in set (0.01 sec)

```
mysql>
```

图 24 账户文件表



```
select CONCAT('*14\r\n','$',LENGTH(redis_cmd),'\r\n',redis_cmd,'\r\n',
'$',LENGTH(redis_key),'\r\n',redis_key,'\r\n',
'$',LENGTH(hkey1),'\r\n',hkey1,'\r\n',
'$',LENGTH(hvalue1),'\r\n',hvalue1,'\r\n',
'$',LENGTH(hkey2),'\r\n',hkey2,'\r\n',
'$',LENGTH(hvalue2),'\r\n',hvalue2,'\r\n',
'$',LENGTH(hkey3),'\r\n',hkey3,'\r\n',
'$',LENGTH(hvalue3),'\r\n',hvalue3,'\r\n',
'$',LENGTH(hkey4),'\r\n',hkey4,'\r\n',
'$',LENGTH(hvalue4),'\r\n',hvalue4,'\r\n',
'$',LENGTH(hkey5),'\r\n',hkey5,'\r\n',
'$',LENGTH(hvalue5),'\r\n',hvalue5,'\r\n',
'$',LENGTH(hkey6),'\r\n',hkey6,'\r\n',
'$',LENGTH(hvalue6),'\r\n',hvalue6,'\r\n')
from (
select 'hset' as redis_cmd,
CONCAT('accountfile:',account) as redis_key,
'status' as hkey1,status as hvalue1,
'opendate' as hkey2,opendate as hvalue2,
'lastexchange' as hkey3,lastexchange as hvalue3,
'balance' as hkey4,balance as hvalue4,
'uid' as hkey5,uid as hvalue5,
'salary' as hkey6,salary as hvalue6
from accountfile
) as c
```

图 25 账户文件表的 sql 语句

```

accountfile:000000406327
accountfile:000002502057
accountfile:000000661878
accountfile:000000088505
accountfile:000000877224
accountfile:000000912563
accountfile:000000887284
accountfile:000000096147
accountfile:000001744728
accountfile:000001441138
accountfile:000000254025
accountfile:000000880171
accountfile:000000073581
accountfile:000000995159
accountfile:000001739744
accountfile:000002680706
accountfile:000001232852
accountfile:000001985051
accountfile:000000445583
accountfile:000000069079
127.0.0.1:6379>

```

图 26 Redis 存储用户信息表

```

127.0.0.1:6379> hget accountfile:000000069079 status
02
127.0.0.1:6379> hget accountfile:000000069079 opendate
1987-01-29
127.0.0.1:6379> hget accountfile:000000069079 lastexchange
1991-12-07
127.0.0.1:6379> hget accountfile:000000069079 balance
0.420
127.0.0.1:6379> hget accountfile:000000069079 uid
1728030
127.0.0.1:6379> hget accountfile:000000069079 salary
0.000
127.0.0.1:6379>

```

图 27 Redis 查询账户文件表

4) 账户流水表的 MySQL 到 Redis 的设计

由于账户流水表没有主键，所以暂时未为其设计 Redis 缓存。

```

mysql> desc accountliquidfile;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| account | char(12) | YES | | NULL | |
| time | int | YES | | NULL | |
| liquidnum | char(9) | YES | | NULL | |
| date | date | YES | | NULL | |
| exchange | float(17,3) | YES | | NULL | |
| balance | float(17,3) | YES | | NULL | |
+-----+-----+-----+-----+-----+-----+
6 rows in set (0.00 sec)

```

图 28 帐户流水表

2.5 主备数据库同步技术

2.5.1 配置主服务器

- 1) 命令: `sudo vim /etc/mysql/mysql.conf.d/mysqld.cnf`, 进入 `mysqld.cnf` 配置文件, 配置如下, 主要是节点 id, 二进制文件名以及大小。

```
[mysqld]
server-id = 1
log-bin = /var/log/mysql/mysql-bin.log
tmpdir = /tmp
binlog_format = ROW
max_binlog_size = 3000M
sync_binlog = 1
expire_logs_days = 7
slow_query_log
pid_file      = /var/run/mysqld/mysqld.pid
socket        = /var/run/mysqld/mysqld.sock
datadir       = /var/lib/mysql
log-error     = /var/log/mysql/error.log
skip-name-resolve
```

- 2) 命令: `sudo systemctl restart mysql`, 重启 mysql, 使配置生效。
- 3) 命令: `create user rpl_user@192.168.198.142 identified by 'xxxxxx';` 创建用户。
- 4) 命令: `grant replication slave on *.* to rpl_user@192.168.198.142;` 给用户授权。
- 5) 命令: `flush privileges;` 刷新。
- 6) 命令: `show grants for replica_user@10.131.35.167;` 查看已授权用户。

```
mysql> show grants for rpl_user@192.168.198.142;
+-----+
| Grants for rpl_user@192.168.198.142 |
+-----+
| GRANT REPLICATION SLAVE ON *.* TO `rpl_user`@`192.168.198.142` |
+-----+
1 row in set (0.00 sec)

mysql> █
```

2.5.2 配置从服务器

- 1) 命令: `sudo vim /etc/mysql/mysql.conf.d/mysqld.cnf`, 进入 `mysqld.cnf` 配置文件, 配置如下, 主要是节点 id, 注意节点不能和主服务器重复, 二进制文件名以及大小。

```
[mysqld]
log_bin = /var/log/mysql/mysql-bin.log
server-id = 2
read_only = 1
tmpdir = /tmp
binlog_format = ROW
max_binlog_size = 3000M
sync_binlog = 1
expire_logs_days = 7
slow_query_log = 1
pid-file = /var/run/mysqld/mysqld.pid
socket = /var/run/mysqld/mysqld.sock
datadir = /var/lib/mysql
log-error = /var/log/mysql/error.log
```

2) 命令: show master status\G。查看主服务器状态。

```
mysql> show master status\G
***** 1. row *****
      File: mysql-bin.000035
      Position: 157
      Binlog_Do_DB:
      Binlog_Ignore_DB:
      Executed_Gtid_Set:
1 row in set (0.00 sec)

mysql> █
```

3) 与主服务器上的二进制文件关联。

```
mysql> CHANGE MASTER TO MASTER_HOST='192.168.198.143',
-> MASTER_USER='rpl_user',
-> MASTER_PASSWORD='qcc216',
-> MASTER_LOG_FILE='mysql-bin.000035',
-> MASTER_LOG_POS=157;
Query OK, 0 rows affected, 8 warnings (0.00 sec)

mysql> █
```

4) 启动从服务器

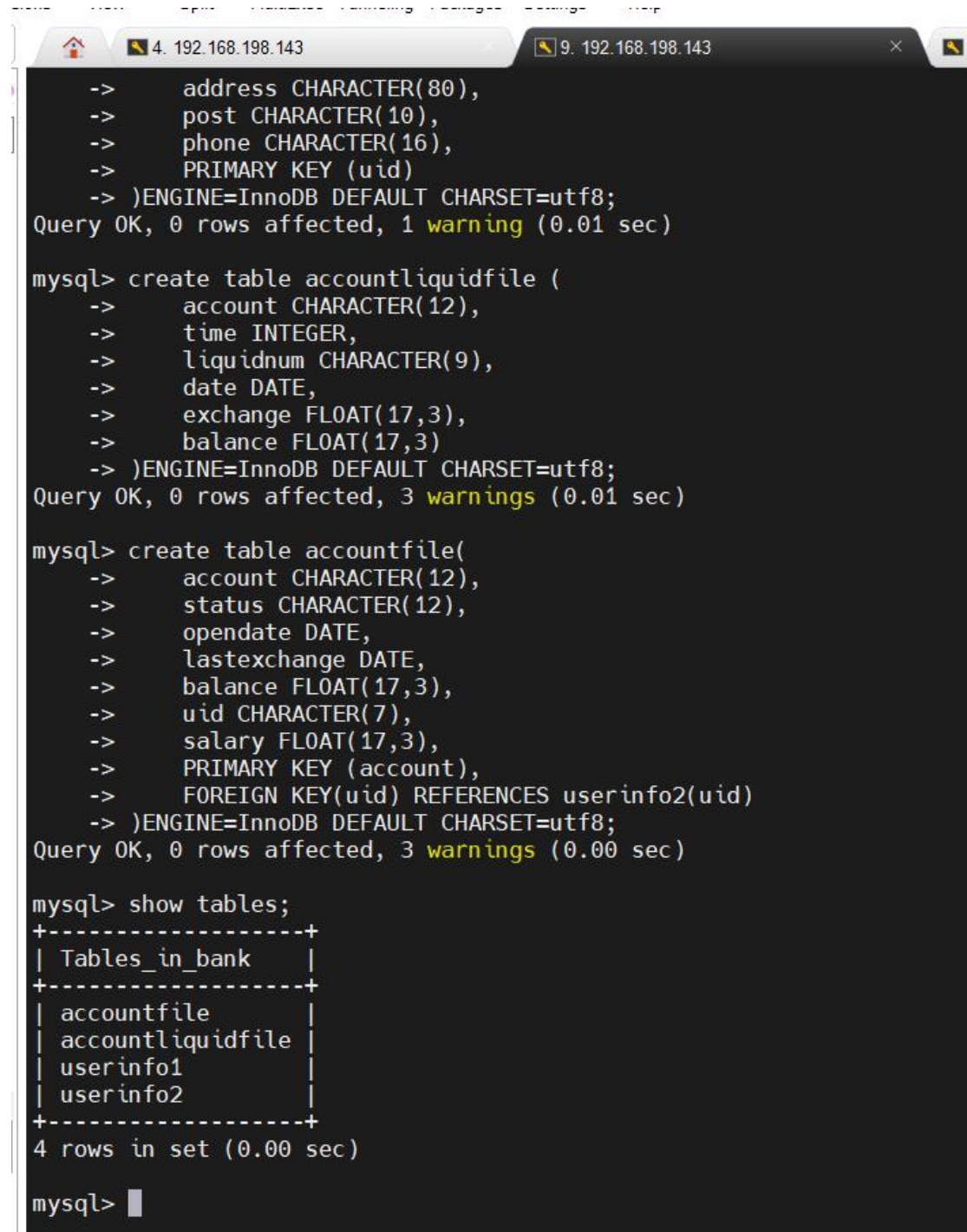
```
mysql> start slave;
Query OK, 0 rows affected, 1 warning (0.01 sec)
```

5) 命令: show slave status\G。查看从服务器状态。

```
mysql> show slave status\G
***** 1. row *****
      Slave_IO_State: Waiting for source to send event
      Master_Host: 192.168.198.143
      Master_User: rpl_user
      Master_Port: 3306
      Connect_Retry: 60
      Master_Log_File: mysql-bin.000035
      Read_Master_Log_Pos: 157
      Relay_Log_File: qcc-virtual-machine-relay-bin.000002
      Relay_Log_Pos: 326
      Relay_Master_Log_File: mysql-bin.000035
      Slave_IO_Running: Yes
      Slave_SQL_Running: Yes
      Replicate_Do_DB:
      Replicate_Ignore_DB:
      Replicate_Do_Table:
```

2.5.3 主备数据库同步

- 1) 主服务器上的 MySQL 中创建数据库 bank，在 bank 中创建用户信息 1 表，用户信息 2 表，账户流水表，账户信息表。可以看到从服务器上也会自动创建数据库 bank，以及四个表。



```
-> address CHARACTER(80),
-> post CHARACTER(10),
-> phone CHARACTER(16),
-> PRIMARY KEY (uid)
-> )ENGINE=InnoDB DEFAULT CHARSET=utf8;
Query OK, 0 rows affected, 1 warning (0.01 sec)

mysql> create table accountliquidfile (
-> account CHARACTER(12),
-> time INTEGER,
-> liquidnum CHARACTER(9),
-> date DATE,
-> exchange FLOAT(17,3),
-> balance FLOAT(17,3)
-> )ENGINE=InnoDB DEFAULT CHARSET=utf8;
Query OK, 0 rows affected, 3 warnings (0.01 sec)

mysql> create table accountfile(
-> account CHARACTER(12),
-> status CHARACTER(12),
-> opendate DATE,
-> lastexchange DATE,
-> balance FLOAT(17,3),
-> uid CHARACTER(7),
-> salary FLOAT(17,3),
-> PRIMARY KEY (account),
-> FOREIGN KEY(uid) REFERENCES userinfo2(uid)
-> )ENGINE=InnoDB DEFAULT CHARSET=utf8;
Query OK, 0 rows affected, 3 warnings (0.00 sec)

mysql> show tables;
+-----+
| Tables_in_bank |
+-----+
| accountfile    |
| accountliquidfile |
| userinfo1      |
| userinfo2      |
+-----+
4 rows in set (0.00 sec)

mysql> █
```

图 29 主数据库


```
mysql> show databases;
+-----+
| Database |
+-----+
| bank     |
| information_schema |
| mysql    |
| performance_schema |
| sys      |
+-----+
5 rows in set (0.01 sec)

mysql> use bank;
Database changed
mysql> show tables;
+-----+
| Tables_in_bank |
+-----+
| accountfile    |
| accountliquidfile |
| userinfo1      |
| userinfo2      |
+-----+
4 rows in set (0.00 sec)

mysql>
```

图 30 从数据库

- 2) 向主数据库中的 userinfo1 表中写入数据，共 303536 条测试数据，可以看到从数据库中也会写入这些数据作为备份。

```
Tools Games Settings Macros Help
sions View Split MultiExec Tunneling Packages Settings Help
4. 192.168.198.143 9. 192.168.198.143 10. 192.168.198.142
9992940 测试专用9992940 测试专用地址9294 00511800 992
9992981 测试专用9992981 测试专用地址9298 00526200 992
9992999 测试专用9992999 测试专用地址9299 00516100 992
9993052 测试专用9993052 测试专用地址9305 00525321 993
9993056 测试专用9993056 测试专用地址9305 00526200 993
9993385 测试专用9993385 测试专用地址9338 00516127 993935637
9993397 测试专用9993397 测试专用地址9339 00516127 993811102
9993609 测试专用9993609 测试专用地址9360 00516100 993
9993747 测试专用9993747 测试专用地址9374 00000000 993
9993778 测试专用9993778 测试专用地址9377 00516100 993
9993996 测试专用9993996 测试专用地址9399 00516123 993
9994076 测试专用9994076 测试专用地址9407 00529800 994213265
9994269 测试专用9994269 测试专用地址9426 00516100 994
9994835 测试专用9994835 测试专用地址9483 00526200 994
9995120 测试专用9995120 测试专用地址9512 00526200 995
9995514 测试专用9995514 测试专用地址9551 00526200 995
9995571 测试专用9995571 测试专用地址9557 00516153 995
9995742 测试专用9995742 测试专用地址9574 00529800 995
9995834 测试专用9995834 测试专用地址9583 00513400 995
9996165 测试专用9996165 测试专用地址9616 00525437 996
9996953 测试专用9996953 测试专用地址9695 00513400 99684
9997000 测试专用9997000 测试专用地址9700 00529800 997
9997226 测试专用9997226 测试专用地址9722 00621000 997
9997913 测试专用9997913 测试专用地址9791 00526100 997
9998297 测试专用9998297 测试专用地址9829 00526100 998
9998326 测试专用9998326 测试专用地址9832 00526200 998
9998486 测试专用9998486 测试专用地址9848 00000000 998
9998487 测试专用9998487 测试专用地址9848 00000000 998
9998488 测试专用9998488 测试专用地址9848 00000000 998
9998489 测试专用9998489 测试专用地址9848 00000000 998
9998495 测试专用9998495 测试专用地址9849 00516100 998723399
9998772 测试专用9998772 测试专用地址9877 00511800 998
9999275 测试专用9999275 测试专用地址9927 00526200 999
9999393 测试专用9999393 测试专用地址9939 00529800 999877953
9999521 测试专用9999521 测试专用地址9952 00526200 999
9999589 测试专用9999589 测试专用地址9958 00526200 999
9999679 测试专用9999679 测试专用地址9967 00529800 999877953
9999975 测试专用9999975 测试专用地址9997 00526200 999
303596 rows in set (0.16 sec)

mysql>
```

图 31 主数据库 userinfo1 表

9992940	测试专用9992940	测试专用地址9294	00511800	992
9992981	测试专用9992981	测试专用地址9298	00526200	992
9992999	测试专用9992999	测试专用地址9299	00516100	992
9993052	测试专用9993052	测试专用地址9305	00525321	993
9993056	测试专用9993056	测试专用地址9305	00526200	993
9993385	测试专用9993385	测试专用地址9338	00516127	993935637
9993397	测试专用9993397	测试专用地址9339	00516127	993811102
9993609	测试专用9993609	测试专用地址9360	00516100	993
9993747	测试专用9993747	测试专用地址9374	00000000	993
9993778	测试专用9993778	测试专用地址9377	00516100	993
9993996	测试专用9993996	测试专用地址9399	00516123	993
9994076	测试专用9994076	测试专用地址9407	00529800	994213265
9994269	测试专用9994269	测试专用地址9426	00516100	994
9994835	测试专用9994835	测试专用地址9483	00526200	994
9995120	测试专用9995120	测试专用地址9512	00526200	995
9995514	测试专用9995514	测试专用地址9551	00526200	995
9995571	测试专用9995571	测试专用地址9557	00516153	995
9995742	测试专用9995742	测试专用地址9574	00529800	995
9995834	测试专用9995834	测试专用地址9583	00513400	995
9996165	测试专用9996165	测试专用地址9616	00525437	996
9996953	测试专用9996953	测试专用地址9695	00513400	99684
9997000	测试专用9997000	测试专用地址9700	00529800	997
9997226	测试专用9997226	测试专用地址9722	00621000	997
9997913	测试专用9997913	测试专用地址9791	00526100	997
9998297	测试专用9998297	测试专用地址9829	00526100	998
9998326	测试专用9998326	测试专用地址9832	00526200	998
9998486	测试专用9998486	测试专用地址9848	00000000	998
9998487	测试专用9998487	测试专用地址9848	00000000	998
9998488	测试专用9998488	测试专用地址9848	00000000	998
9998489	测试专用9998489	测试专用地址9848	00000000	998
9998495	测试专用9998495	测试专用地址9849	00516100	998723399
9998772	测试专用9998772	测试专用地址9877	00511800	998
9999275	测试专用9999275	测试专用地址9927	00526200	999
9999393	测试专用9999393	测试专用地址9939	00529800	999877953
9999521	测试专用9999521	测试专用地址9952	00526200	999
9999589	测试专用9999589	测试专用地址9958	00526200	999
9999679	测试专用9999679	测试专用地址9967	00529800	999877953
9999975	测试专用9999975	测试专用地址9997	00526200	999

303596 rows in set (0.16 sec)

mysql>

图 32 从数据库 userinfo1 表

2.6 xml 配置文件的创建与读取

为了提高银行交易系统的交互性，我们将数据库用户及密码等配置文件信息通过 xml 文件进行配置和读取。

2.6.1 下载并配置 libxml2 库

通过命令行：`sudo apt-get install libxml2-dev`
`sudo apt-get install libxml2`

安装 libxml2 库到默认路径/user/include/下，通过命令行：`dpkg -s libxml2-dev`可以检查安装包信息，如下图所示完成安装。


```
1 Package: libxml2-dev
2 Status: install ok installed
3 Priority: optional
4 Section: libdevel
5 Installed-Size: 2862
6 Maintainer: Ubuntu Developers <ubuntu-devel-discuss@lists.ubuntu.com>
7 Architecture: amd64
8 Multi-Arch: same
9 Source: libxml2
10 Version: 2.9.1+dfsg1-3ubuntu4.12
11 Depends: libxml2 (= 2.9.1+dfsg1-3ubuntu4.12)
12 Suggests: pkg-config
13 Description: Development files for the GNOME XML library
14 XML is a metalanguage to let you design your own markup language.
15 A regular markup language defines a way to describe information in
16 a certain class of documents (eg HTML). XML lets you define your
17 own customized markup languages for many classes of document. It
18 can do this because it's written in SGML, the international standard
19 metalanguage for markup languages.
20 .
21 Install this package if you wish to develop your own programs using
22 the GNOME XML library.
23 Homepage: http://xmlsoft.org/
```

图 33 libxml2 库信息

2.6.2 创建 xml 配置文件

以存储数据库用户名及密码为例，我们通过以下流程及程序创建出用户配置文件如下图 4 所示。



图 34 创建 xml 配置文件流程

```

//创建phone_books
static int create_phone_books(const char *phone_book_file)
{
    assert(phone_book_file);

    xmlDocPtr doc = NULL;
    xmlNodePtr root_node = NULL;

    //创建一个xml 文档
    doc = xmlNewDoc(BAD_CAST"1.0");
    if (doc == NULL) {
        fprintf(stderr, "Failed to new doc.\n");
        return -1;
    }

    //创建根节点
    root_node = xmlNewNode(NULL, BAD_CAST"phone_books");
    if (root_node == NULL) {
        fprintf(stderr, "Failed to new root node.\n");
        goto FAILED;
    }

    //将根节点添加到文档中
    xmlDocSetRootElement(doc, root_node);

    if (add_phone_node_to_root(root_node) != 0) {
        fprintf(stderr, "Failed to add a new phone node.\n");
        goto FAILED;
    }

    //将文档保存到文件中，按照utf-8编码格式保存
    xmlSaveFormatFileEnc(phone_book_file, doc, "UTF-8", 1);
    //xmlSaveFile("test.xml", doc);
    xmlFreeDoc(doc);

    return 0;
FAILED:
    if (doc) {
        xmlFreeDoc(doc);
    }

    return -1;
}

```

图 35 生成配置文件核心代码

```

phone_book.xml
1  <?xml version="1.0" encoding="UTF-8"?>
2  <phone_books>
3  <phone id="1">
4      <name>zxx</name>
5      <password>66666666</password>
6  </phone>
7  </phone_books>
8

```

图 36 用户配置文件

2.6.3 解析用户配置文件

我们生成用户配置文件后，在程序中我们需要对配置文件进行解析，我们按照如下流程，就可读取用户配置文件如下图 6 所示。



图 37 解析 xml 配置文件流程图

```

static int parse_phone_book(const char *file_name)
{
    assert(file_name);

    xmlDocPtr doc;    //xml整个文档的树形结构
    xmlNodePtr cur;    //xml节点
    xmlChar *id;        //phone id

    //获取树形结构
    doc = xmlParseFile(file_name);
    if (doc == NULL) {
        fprintf(stderr, "Failed to parse xml file:%s\n", file_name);
        goto FAILED;
    }

    //获取根节点
    cur = xmlDocGetRootElement(doc);
    if (cur == NULL) {
        fprintf(stderr, "Root is empty.\n");
        goto FAILED;
    }

    if (!xmlStrcmp(cur->name, (const xmlChar *)"phone_books")) {
        fprintf(stderr, "The root is not phone_books.\n");
        goto FAILED;
    }

    //遍历处理根节点的每一个子节点
    cur = cur->xmlChildrenNode;
    while (cur != NULL) {
        if (!xmlStrcmp(cur->name, (const xmlChar *)"phone")) {
            id = xmlGetProp(cur, "id");
            printf("id:%s\t", id);
            parse_phone(doc, cur);
        }
        cur = cur->next;
    }
    xmlFreeDoc(doc);
    return 0;
}
FAILED:
    if (doc) {
        xmlFreeDoc(doc);
    }
    return -1;
}

```

图 38 解析 xml 配置文件核心代码

```

zxx@gxy:~/ddd$ gcc -I/usr/include/libxml2 readxml.c -o readxml -lxml2
zxx@gxy:~/ddd$ ./readxml
id:1    name: zxx    password: 66666666    zxx@gxy:~/ddd$

```

图 39 解析 xml 配置文件结果