MySQL+Redis 数据同步方法对比总结

为什么使用 MySQL+Redis？
分工操作，取各自所长，效率高！

MySQL 和 Redis 对比:

MySQL 的优点：

1>.支持关系型数据
2>.支持事务
3>.数据可持久存储
4>.存储数据量大
5>.可处理复杂业务逻辑

MySQL 的缺点：

1>.执行需要准备/解析/优化/执行，步骤多，效率慢
2>.消耗资源多

Redis 优点：

1>.速度快
2>.消耗资源少
3>.支持高并发
4>.逻辑简单

Redis 缺点:

1>.对事务支持不好
2>.处理关系数据困难
3>.持久化存储困难

MySQL+Redis 实现难点：

1>.有些业务不能使用 Redis 简单逻辑处理。
2>.对于需要高并发访问，Mysql 和 Redis 之间的数据变更同步问题。

解决办法：

1>.业务动静数据分离，需要高并发的部分业务数据同步到 Redis，应用访问 Redis 进行业务数据处理，例如：砍价功能数据。
2>.通过以下方法实现数据同步：

MySQL 和 Redis 之间数据同步实现：

注意:本文档只给出大体的实现方法，由于各个业务情况不同，具体逻辑根据实际情况而定，不能一一分析全面。例如：读 redis 写 MySQL 这样的逻辑和细节，不做具体分析。

1. **在程序里实现 MySQL+Redis 的同步。**

实现说明：

1>.应用程序进行查询数据，如果没有查询到，就去对方把数据拉过来。
2>.应用在写数据的时候两边都写。
3>.应用在读数据的时候读一边。

下面是简单的逻辑代码实现：

cat mytoredis.py

```
#!/usr/bin/python
import MySQLdb
```

```
import redis

col_name=['id','name','sal','token']
def select(sql):
    db = MySQLdb.connect("127.0.0.1","test","123456","test",3306 )
    cursor = db.cursor()
    cursor.execute(sql)
    results = cursor.fetchall()
    db.close()
    return results

def get_v(tab_name,col_name,id):
   r = redis.Redis(host='127.0.0.1',port=6379,db=0)
   rr = r.hget(tab_name+':'+col_name[0]+':'+str(id),col_name)
   cols_len = len(col_name)
   print('redis:'+str(rr))
   if not rr :
      sql="select * from rd_test where id = %d" %id
      res=select(sql)
      print('sql:'+str(res))
      if res :
         for row in res:
            for col in range(cols_len):
               r.hset(tab_name+':'+col_name[0]+':'+str(id),col_name[col],row[col])

get_v('rd_test',col_name,4)
```

cat redistomy.py

```
#!/usr/bin/python
import MySQLdb
import redis
import time

rec_tim = 22222

col_name=['id','name','sal','token']

def get_redis(tab_name,col_name,id,rec_tim):
   r = redis.Redis(host='127.0.0.1',port=6379,db=0)
   tim = r.hget(tab_name+':'+col_name[0]+':'+str(id),'tt')
   if int(tim) > rec_tim :
      rec_tim = tim
      rr = r.hmget(tab_name+':'+col_name[0]+':'+str(id),col_name)
   return rr
```

```
    def in_my(tab_name,col_name,id,rec_tim):
        db = MySQLdb.connect("127.0.0.1","test","123456","test",3306 )
        cursor = db.cursor()
        rows=get_redis('rd_test',col_name,id,rec_tim)
        print(rows)
        if rows :
           cols=str(rows).replace('[','').replace(']','').replace('None','''')
           sql1 = " delete from %s where %s = %s" % (tab_name,col_name[0],id)
           re=cursor.execute(sql1)
           sql2 = "insert into %s values(%s)" % (tab_name,cols)
           re=cursor.execute(sql2)
    db.commit()
        db.close()

in_my('rd_test',col_name,6,rec_tim)
```

2. 通过 Pymysqlreplication 的 python 第三方库方法实现 MySQL 和 Redis 同步

实现原理：

1>.通过 python 的 pymysqlreplication 第三方库抓取 MySQL 的 binlog 日志。

2>.然后把解析出来的数据同步到 Redis 上。

下面是简单实现抓取 MySQL 的 binlog 数据：

```
    #!/usr/bin/env python
     # -*- coding: utf-8 -*-

    from pymysqlreplication import BinLogStreamReader
    from pymysqlreplication.row_event import (
            DeleteRowsEvent,
            UpdateRowsEvent,
            WriteRowsEvent,
     )
    import sys
    import json

    def main():
            mysql_settings = {'host': '127.0.0.1',
                                      'port': 3306, 'user': 'root', 'passwd': '123456'}
            stream = BinLogStreamReader(
                  connection_settings=mysql_settings,
                  server_id=2,
                  blocking=True,
                  only_schemas=['test'],
                  only_events=[DeleteRowsEvent, WriteRowsEvent, UpdateRowsEvent],
```

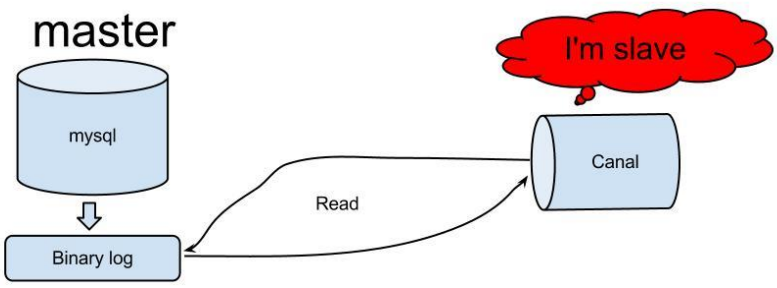
```
                    resume_stream=True,
                    log_file='mysql-bin.000022', log_pos=192)

            for binlogevent in stream:
                for row in binlogevent.rows:
                    event = {"schema": binlogevent.schema, "table": binlogevent.table,
"log_pos": binlogevent.packet.log_pos}
                    if isinstance(binlogevent, DeleteRowsEvent):
                        event["action"] = "delete"
                        event["values"] = dict(row["values"].items())
                        event = dict(event.items())
                    elif isinstance(binlogevent, UpdateRowsEvent):
                        event["action"] = "update"
                        event["before_values"] = dict(row["before_values"].items())
                        event["after_values"] = dict(row["after_values"].items())
                        event = dict(event.items())
                    elif isinstance(binlogevent, WriteRowsEvent):
                        event["action"] = "insert"
                        event["values"] = dict(row["values"].items())
                        event = dict(event.items())
                    print json.dumps(event)
                    sys.stdout.flush()

            stream.close()


    if __name__ == "__main__":
        main()
```

输出格式：

{"action": "insert", "table": "yy", "log_pos": 5876, "values": {"iD": 14}, "schema": "test"}

3. **Canal+写程序实现 MySQL+Redis 的同步。**

实现原理：



1>.canal 模拟 mysql slave 的交互协议，伪装自己为 mysql slave，向 mysql master 发送 dump 协议

2>.mysql master 收到 dump 请求，开始推送 binary log 给 slave(也就是 canal)

3>.canal 解析 binary log 对象(原始为 byte 流)

4>.然后把解析出来的数据同步到 Redis 上。

输出格式：


```
    {
     "binlog": "mysql-bin.000017:24777",
     "db": "test",
     "table": "load_test",
     "eventType": "INSERT",
     "before": "",
     "after": {
          "AUTO_INCREMENT": "1",
          "AVG_ROW_LENGTH": "0",
          "CHECKSUM": "",
          "CHECK_TIME": "",
          "CREATE_OPTIONS": "",
          "CREATE_TIME": "2018-07-05 16:15:38",
          "DATA_FREE": "0",
          "DATA_LENGTH": "16384",
          "ENGINE": "InnoDB",
          "INDEX_LENGTH": "32768",
          "MAX_DATA_LENGTH": "0",
          "ROW_FORMAT": "Compact",
          "TABLE_CATALOG": "def",
          "TABLE_COLLATION": "latin1_swedish_ci",
          "TABLE_COMMENT": "",
          "TABLE_NAME": "pp",
          "TABLE_ROWS": "0",
          "TABLE_SCHEMA": "test",
          "TABLE_TYPE": "BASE TABLE",
          "UPDATE_TIME": "",
          "VERSION": "10"
     },
     "time": "2018-09-03 11:06:29"
}
```


4. Gearman+udf 插件 +Trigger+写程序实现 MySQL+Redis 的同步

实现原理：

1>.创建插件函数，功能是把数据发送给 gearman

2>.创建表的触发器，当表数据有变更，就把变更数据通过函数发送给 gearman

3>.gearman 接收到数据，然后把数据同步到 redis

数据输出:

"{\"id\":1,\"volume\":\"aaaaa\"}"

对比以上 4 种方法的优缺点：

方法一:业务程序实现 方法二:pymysqlreplication 实现 方法三:canal 实现 方法四:gearman 实现

优点：

方法一:完全由业务层实现，业务开发最了解数据的变更，可以根据具体数据变更进行数据同步操作，数据一致性较高。

方法二:通过 binlog 日志分析数据，对 MySQL 的性能影响较小。

方法三: 通过 binlog 日志分析数据，对 MySQL 的性能影响较小适合大数量数据同步。

方法四:通过触发器获取数据变更，数据实时性同步较高。

缺点:

方法一:影响业务层业务处理速度。

方法二:数据同步实时性较低。

方法三: 数据同步实时性较低。

方法四:对 MySQL 数据库性能影响较大。

最后给出建议：

1>.当需要同步的数据量很大并且实时性要求不是很高时，使用方法三，因为阿里云的 DTS 数据迁移消息队列就是用此程序，经过测试，可用性高。

2>.当需要实时性很高，需要同步的数据量很少时，可以使用方法四，通过触发器处理数据同步速度快。

3>.对于不想安装其他程序，又想实现数据同步，就可以使用程序实现。

具体情况根据实际需求而定,这里只给出参考意见。