DBProxy 后端 backend 状态检测设计文档

变更说明

日期	版本	变更位置	变更说明	作者
2013-04-10	0.0.1		创建初始文档	谢文
2013-06-20	0.2.0		设计细化	侯金轩

本文档和最新代码不一定是严格匹配，仅仅作为参考

目录

1 目标

更改 MysqlProxy 的源代码后端数据库健康检查模块，为其增加执行外部命令的能力。将数据库状态检查的主要逻辑放到外部自定义的脚本中，检查模块调用外部脚本，根据其返回值取得数据库状态。

2 系统概要设计

从 1 中我们了解到，Backend 后端检测模块主要功能包括 zabbix agent 的功能函数调用及结果分析、backend 状态的更新等。此外，调度的方便我们会为每个 backend 起一个检测的线程。综上考虑，后端检测模块从功能上划分为三个模块：1. 后端 zabbix 命令调用及通信模块；2. 后端检测触发调度线程框架；3. backend 状态更新及后续处理模块。下面分别对模块做简单介绍：

2.1 后端 zabbix 命令调用及通信模块

后端 zabbix 命令调用及通信模块：负责实现与后端 zabbix agentd 的连接建立、命令发送、返回结果处理并向上层返回 backend 的状态。该模块会根据与 zabbix agentd 约定的方式将从 zabbix agent 转换成我们 dbproxy 关心的状态返回。

DBproxy 与 zabbix agentd 之间连接的建立、命令的发送及结果集的接受（当然包括超时的处理、连接的自己的断开）可以通过基于 libevent 结合状态机的形式实现异步非阻塞的 io。当然也可以同步方式实现。

数据库状态判断灵活性方面的考虑，dbproxy 其对外暴露脚本调用的接口。参数至少包括检测数据库的用户名、密码、被检测状态的数据库实例的 ip 和端口信息。原先的主库地址、主库 port 等信息暂时不做考虑。

2.2 后端检测触发调度框架

后端检测触发调度框架：负责对后端状态检测的触发调度。现在有四种个试选方案供选择：

2.2.1 类似后端连接的异步建立，检测的过程中的等待事件还是通过 mysql-proxy 原来的 socket pair 的由原来的处理线程处理。（不足是怕前端线程处理 mysql 查询请求压力太大，后端检测不能及时触发。优点简单，这部分代码不需要在自己写）

2.2.2 仿照 mysql-proxy 的多线程处理方式，再写一套线程池来专门处理后端的通信（不足：惊群）

2.2.3 同样是仿照 mysql-proxy 的多线程处理方式，只不过是一个 socket 有专门的线程处理。（同样的是临时的时间注册，但是在 zabbix_socket 里面需要存放相应的处理线程的

event_base）

2.2.4 针对每个 backend 单独启动一个线程负责检测该 backend 的状态，即一个 backend 由一个特定的线程来负责。不需要线程的复用，这样做可能会简单些，出错的概率较低。大家比较推荐这种方式实现。

2.3 状态更新及后续处理模块

状态更新及后续处理模块：负责处理 backend 状态更新及善后工作。如状态改变时，对 backend 上面连接的处理、负载均衡依赖数据的更新等等。

3 runcmd 概要设计

3.1 runcmd 后台服务（后端 zabbix 命令调用及通信模块）

MySQLProxy 本身是多线程的服务，多线程程序执行外部命令需要考虑死锁和内存资源泄露等问题，通常不建议这么做，实现起来也较复杂。为了保证 MySQLProxy 服务的健壮性，因此在 MySQLProxy 之外单独起一个 runcmd 后台守护进程专门提供执行外部命令的服务，MySQLProxy 作为客户端与该服务交互，向后台服务发送要执行的命令，后台服务端接收到客户端请求后，创建一个子进程处理请求执行命令，执行完后将执行结果返回给客户端。方便简单起见，直接采用 zabbix agentd 作为后台服务程序。但是据我了解现在组内对 zabbix agentd 部署位置存在认识上面的误解（这里大家现在认识比较一致了：zabbix agentd 和 dbproxy 部署在一起看成 dbproxy 的组件一块发布）。

编码参考：runcmd 服务用 zabbix_agentd，需了解其通信协议

3.2 mysqlproxy 健康检查模块（状态更新及后续处理模块）

MySQLProxy 健康检查模块当检查某一备库状态时，将其 IP 地址和端口等信息拼出要执行的命令字符串，发送给 runcmd 服务端，等待服务器返回执行结果，根据返回值得到数据库健康状态。

编码参考：参考 zabbix_get 客户端的实现，了解其协议。可用 libevent 及其定时器实现客户端和实现超时

注意：dbproxy 与 zabbix 通信获取后端 backend 的状态时，会得到两类结果。1. 按照约定的形式返回实例的状态，这种可以比较确定的判定被检测实例的状态；2. 由于网络原因、zabbix 意外 down 机或其他原因导致的 dbproxy 与 zabbix agentd 间 socket 无法正确建立或者是 dbproxy 与 zabbix agentd 通讯超时或者是 dbproxy 从 zabbix agentd 获取到结果不符合要求的情况。情况 1 可以比较容易的判断；情况 2，属于业务规则问题，片中可用性考虑

倾向于多试几次，若都是无法处理则置 down。

3.3 备库检查脚本

完成数据库备库状态检查。主要操作是连接数据库、查看备库状态、返回备库状态

编码参考：shell 或 perl 等脚本语言实现。注意 zabbix_agentd 没有取脚本返回值，所以不能像 haproxy 那样直接判断返回值，需要脚本将返回值按一定格式写到标准输出里

4 模块详细设计

4.1 runcmd 后台服务（后端 zabbix 命令调用及通信模块）

4.1.1 接口设计

Runcmd 后台服务用 zabbix_agentd，自定义一些与检查脚本对应的检查项(item)。检查模块通过 zabbix_agentd 调用检查脚本，通信协议基于 zabbix_agentd 协议，具体格式见后面模块。

检查项配置文件：
/etc/zabbix/zabbix_agentd.d/userparameter_mysqlproxy.conf

配置：
UserParameter=mysqlproxy.backends_status[*],/home/mysql/dbadmin/runcmd/check_mysqlproxy_backends.sh “$1” “$2” “$3” “$4” “$5”

注意：除了 dbproxy 与 zabbix_agentd 的通信协议外，还需要与 dbproxy 协商好脚本具体的返回格式及相关字段的含义（~~下面会有说明~~推荐使用 errno=1;errmsg=”test”;status=down 这种形式）。

4.2 mysqlproxy 健康检查模块(状态更新及后续处理模块)

该模块通过建立与 zabbix_agented 的 socket 连接，接着向其按照固定的格式发送三个 sokect 数据包（head 数据包、命令包长数据包、命令数据包）；然后等待并接受 zabbix_agentd 返回的三个 socket 数据包（head 数据包、返回信息包长数据包、返回信息数据包）。注意需要对返回失败、ZBX_NOTSUPPORTED、返回超时等特殊的返回进行处理。

4.2.1 接口设计

检查模块向 zabbix_agentd 发请求，并接收从 zabbix_agentd 返回的脚本执行结果，取得返回值，根据返回值修改备库状态。

请求消息体的格式参考 zabbix_get，如下：
mysqlproxy.backends_status[IP 地址,端口,用户名,密码,UNIX 套接字,主库 IP 地址,主库端口]\n
参数里不能带逗号

响应消息体的格式是键值对，如下：
errno=错误号
errmsg=错误信息
dbrole=角色

errno=0 表示备库状态正常，其它非 0 值表示备库状态异常(连不上/延迟大)，errmsg 是补充信息

4.2.2重要数据结构

Zabbix_socket:实现对 dbproxy 与 zabbix_agent socket 连接的封装。需要有文件描述符、socket 在数据传输中的状态、及注册事件。如下：

```
typedef struct {
    int fd;                     /**< socket-fd */
    struct event event; /**< events for this fd */

    network_address *src; /**< getsockname() */
    network_address *dst; /**< getpeername() */

    enum zabbix_trans_state state;
} zabbix_socket;
```

4.2.3重点函数说明

zabbix_agent_connect(const char *source_ip，guint16 port，gint timeout):建立于 zabbix_agentd 的连接；
zabbix_agent_write_header(zabbix_socket zabbix_agent):向 zabbix_agentd 发送 zabbix 的头数据包；
zabbix_agent_write_length(zabbix_socket zabbix_agent, zabbix_socket):向 zabbix_agentd 端发送信息包的数据长度；
zabbix_agent_write_message(zabbix_socket zabbix_agent,const char *msg_buffer, guint len):向 zabbix_agentd 端发送指定长度的字符串；
zabbix_agent_read_header(zabbix_socket zabbix_agent):从 zabbix_agent 端读取 zabbix_header

数据
zabbix_agent_read_length(zabbix_socket zabbix_agent):从 zabbix_agent 端读取后续信息的长度
zabbix_agent_read_message(zabbix_socket zabbix_agent, char*buffer, guint *len): 从 zabbix_agent 端读取相应的脚本的返回结果即上述定义的键值对。

zabbix_agent_close(zabbix_socket zabbix_agent):关闭与 zabbix_agent 的连接

4.3 备库检查脚本

4.3.1 接口设计

脚本输入参数按固定位置排列，依次是：IP 地址，端口，用户名，密码，UNIX 套接字，主库 IP 地址，主库端口

IP 地址：可空，默认 127.0.0.1
端口：可空，默认 3306
用户名：可空
密码：可空
UNIX 套接字：可空
主库 IP 地址：可空
主库端口：可空

脚本输出是键值对格式，如：
errno=错误号
errmsg=错误信息
dbrole=角色

举例如下：

1. 数据库连接异常
errno=1
errmsg=db connection failed

2. 主库正常
errno=0
dbrole=rw

3. 备库复制线程状态异常
errno=2

errmsg=slave stopped
dbrole=ro

4. 备库连接的主库地址不对
errno=4
errmsg=slave connected to wrong master
dbrole=ro

5. 备库复制延迟
errno=3
errmsg=slave lagging behind master
dbrole=ro

6. 备库状态正常
errno=0
dbrole=ro
（注意脚本的功能是判定实例的死活，检测时脚本可以返回 errno：上述设计完全是根据 errno 来判定实例的死活的；errmsg：使得错误信息更明确；status 或者是 dbrole 可以根据需要增加）

5 重要逻辑流程

5.1 重要逻辑分析

5.1.1 健康检查相关逻辑

5.1.1.1 功能描述

为了减少频繁后端检测的情况下线程频繁建立和销毁的代价，我们后端检测采用线程池的方式。因而会采用异步的方式，状态机包括与 zabbix 通信的六个状态。

与此相关的逻辑包括：1. 以异步方式轮询检查后端数据库；2. 发请求：连接(短连接)zabbix_agentd，生成请求消息包，发送请求；3.处理响应：接收响应，断开连接，分析返回值，修改状态。

5.1.1.2 具体流程分析

1. 找到 RW 角色的数据库，设置为当前主库地址端口。如有多个则报错，sleep 一会儿，等下一轮检查
2. 开始检查一个数据库
3. 连接 zabbix_agentd，如失败，则报错，检查下一个库
4. 生成请求消息
5. 发送请求，如失败，则报错，检查下一个库
6. 等待接收响应，如失败，则报错，检查下一个库
7. 断开连接
8. 处理响应，分析返回的结果
9. 如返回 NOT_SUPPORT，则报错，检查下一个库
10. 取出返回的 errno 和 dbrole
11. 根据 dbrole 修改数据库角色，和当前主库地址端口
12. 根据 errno 修改数据库状态
13. 开始检查下一个后端数据库状态
14. 都查完了，sleep 一会儿，等下一轮检查

通过跟踪 zabbix_server 与 zabbix_agent 通信的协议，dbproxy 与 zabbix_agentd 的交互涉及到六个阶段。
阶段 1：dbproxy 调用 zabbix_agent_write_header(zabbix_socket zabbix_agent)向 zabbix_agentd 发送包头数据包；
阶段 2：dbproxy 调用 zabbix_agent_write_length(zabbix_socket zabbix_agent, guint len)向 zabbix_agentd 发送命令数据的长度（注意为 8 个字节）
阶段 3：dbproxy 调用 zabbix_agent_write_message(zabbix_socket zabbix_agent,const char *msg_buffer, guint len):向 zabbix_agentd 端发送指定状态监测命令
阶段 4：zabbix_agent_read_header(zabbix_socket zabbix_agent): 从 zabbix_agent 端读取 zabbix_header 数据
阶段 5：proxy 调用 zabbix_agent_read_length(zabbix_socket zabbix_agent):从 zabbix_agent 端读取后续信息的长度
阶段 6：zabbix_agent_read_message(zabbix_socket zabbix_agent, char*buffer, guint *len):从 zabbix_agent 端读取相应的脚本的返回结果。

5.1.1.3 实现过程中注意事项

1. 读取及写入借助 libevent 实现异步非阻塞的方式；
2. 注意会存在 zabbix_agentd 没有按照预期到来的情形；（包括超时、命令不存在时。需要在状态机中考虑到这些特殊的情况）
3. 多线程的逻辑打算使用 libevent 实现，定时的线程为主线程负责向后端的 worker 线程分发事件，worker 处理 dbproxy 与 zabbix_agentd 的通信的流程
4. 在主库 down 掉时，需要将写服务关闭。合适开启？
5. 主库 down 掉时，需要重新选主。原来主库上面的连接如何处理是最大的问题。
6. 在 dbproxy 刚启动时，需要对 backend 的存活状态和主备关系进行初始化？

5.1.2 检查脚本相关逻辑

5.1.2.1 功能描述

与此相关的逻辑包括：1. 连接数据库；2. 查询备库状态；3.输出检查结果

5.1.2.2 具体流程分析

1. 用输入的参数（IP 地址，端口，用户名，密码，UNIX 套接字）连接数据库
2. 查看数据库是否只读(show global variables like ‘read_only’)
3. 若非只读，则认为是主库，输出结果，退出。否则继续
4. 查看备库状态(show slave status)
5. 若复制线程停止，则输出结果，退出
6. 若主库地址不正确，则输出结果，退出
7. 若复制有延迟，则输出结果，退出
8. 说明备库正常，输出结果，退出

6 难点问题分析

上面已经对 dbproxy 后端检测的逻辑做了分析设计。但是在获取到后端的 backend 的状态之后如何精巧的更新 backend 的状态及相应的 backend 节点上面的连接如何比较优雅的处理掉?

1. Dbproxy 触发后端检测的框架？
   之前的是检测一个后端就新启动一个线程，检测完毕后线程关闭。这样的话会存在一个问题，就是若检测的比较频繁的话 dbproxy 会频繁的创建线程。Cpu 资源会不会大量的消耗？

   现在打算采用线程池的方式实现后端的检测，一组 worker 线程组成线程池负责对后端实现检测。Worker 线程池同时监听一个 socket pair,检测调度线程或其他管理线程负责向 socket pair 中添加事件，worker 线程负责去获取事件，并注册在自己的 event_base 上面并异步处理与 zabbix_agentd 之间的 socket 的状态。继续注册事件是使用 mysql_proxy 前面线程采用的类似惊群的方式还是一个线程拿到一个 socket 后会处理该 socket 所有的状态？两种方式都可以，甚至是可以采用类似异步建立连接的方式直接使用前面建立的 event_thread 服务线程处理与 zabbix_agentd 的异步通信。

2. 拿到某个后端的状态后如何操作？
   只有在状态有改变的情况下，才需要做特殊考虑。若检测的后端的死活状态与上一次的

死活状态一致，不需要做特殊处理，对该后端的检测成功推出即可。

当检测的后端状态改变时：

1. Up→Down
   A. 若是主库
      需要关闭 dbproxy 对外提供的对外写服务；
      何时重新选主最早至少应该在本轮检测完成时吧？由于后端状态检测是异步完成的，需要设置个标志位是否检查完全？（但是只有检测主库状态的线程才知道主库 down 的消息，他应该触发重新选主。等待至状态检测完毕？）
   B. 若是备库
      更新状态、然后对其上的连接做处理。

2. Down→Up
   更新该后端的状态，为可用。

注意：上面的两个状态的变化、都需要考虑对负载均衡算法的影响。特别是负载均衡算法为加权轮询的时候。

3. 若后端 backend 是物理意义上面的 down 机，则其上面的连接会自动的取消掉。

4. 若后端 backend 只是逻辑意义上面的 down，该怎么办呢？
   2.1 用户的连接可能处于状态机的各个状态：语句已经执行或没有执行、结果返回给了 client 或没有返回给 client 或这只是返回了一半的结果、con 的连接可能是被缓存着的？请求是否重试？
   2.2 由于是第三方的线程来更新 backend 的状态，如何通知到哪些连接处理的线程（并且使用的是该 backend 上面的连接），会产生什么样的影响？
   A. 对于连接池里面缓存的连接，这个可以比较轻松的处理（可以直接杀掉，要注意多线程同步的问题）。
   B. 对于 client 在使用的连接中的连接如何处理？比较棘手
   主要考虑到以下问题：
   a. 考虑到易用性，需要向 client 端返回错误代码。但是 client 是不会被动接受 dbproxy 的数据的，只有在 read_query_result 和向 client 返回完整的结果集之前，client 可以实时接受 dbproxy 数据。
   b. 何时向 client 返回错误的代码信息。同样难题是 client 不会被动的接受 dbproxy 的数据、再者是不确定在返回结果返回一半的情况下在返回 client 错误信息是否会正常处理。可选的方案：1.等待进行中的请求处理完成，连接下一次发送过来请求数据时（这样子好像是有点延迟）？2.第三方线程实时的将此类连接关闭（好像是不太友好），需要考虑到线程的同步。
   c. 如果是主库 down，其上的连接是不是立即释放？不能等待下次连接请求执行结束再关闭？这样会对上千个并发的连接加锁，从中选择出主库上面的连接 kill?

注意：对于 down 掉的 backend 上面已存在的连接，考虑到业务的复杂性。我们倾向于直接将上面的 client 的连接关闭。（模拟 mysqld 关闭会关闭 client 的连接一样）

7 参考文献