目的：

1. 确认在slave启动后，同步的第一批binlog(从故障点到当前的积压日志)，此时的模式是半同步还是异步获取日志?
2. 当Master Rpl_semi_sync_master_status=OFF 时， 是否连接上的所有slave都是异步复制。
3. 同1, 当Master Rpl_semi_sync_master_status=ON 时， 连接上来的是异步复制，需要追上积压日志后会自动转为半同步复制?
4. 首次连接按位点获取日志时，这个位点从哪里取？（待定）
5. Master根据什么信号判断连接过来的slave是采用半同步模式还是异步模式?

Master Status:

```
mysql> show global status like '%semi_sync%';
+--------------------------------------------+-------+
| Variable_name                              | Value |
+--------------------------------------------+-------+
| Rpl_semi_sync_master_clients               | 1     |
| Rpl_semi_sync_master_net_avg_wait_time     | 0     |
| Rpl_semi_sync_master_net_wait_time         | 0     |
| Rpl_semi_sync_master_net_waits             | 0     |
| Rpl_semi_sync_master_no_times              | 0     |
| Rpl_semi_sync_master_no_tx                 | 0     |
| Rpl_semi_sync_master_status                | ON    | <<<
| Rpl_semi_sync_master_timefunc_failures     | 0     |
| Rpl_semi_sync_master_tx_avg_wait_time      | 0     |
| Rpl_semi_sync_master_tx_wait_time          | 0     |
| Rpl_semi_sync_master_tx_waits              | 0     |
| Rpl_semi_sync_master_wait_pos_backtraverse | 0     |
| Rpl_semi_sync_master_wait_sessions         | 0     |
| Rpl_semi_sync_master_yes_tx                | 0     |
+--------------------------------------------+-------+
14 rows in set (0.01 sec)

mysql> show variables like '%semi_sync%';
+-------------------------------------------+------------+
| Variable_name                             | Value      |
+-------------------------------------------+------------+
| rpl_semi_sync_master_enabled              | ON         | <<< 前提条件
| rpl_semi_sync_master_timeout              | 10000      |
| rpl_semi_sync_master_trace_level          | 32         |
| rpl_semi_sync_master_wait_for_slave_count | 1          |
| rpl_semi_sync_master_wait_no_slave        | ON         |
| rpl_semi_sync_master_wait_point           | AFTER_SYNC |
+-------------------------------------------+------------+
6 rows in set (0.01 sec)
```

2. 当Master Rpl_semi_sync_master_status=OFF 时， 是否连接上的所有slave都是异步复制。 YES

```
.\mysql-8.0.39\plugin\semisync\semisync_source.cc

int ReplSemiSyncMaster::writeTranxInBinlog(const char *log_file_name,
                                           my_off_t log_file_pos) {

...

if (is_on()) {
    assert(active_tranxs_ != nullptr);
    if (active_tranxs_->insert_tranx_node(log_file_name, log_file_pos)) {
      /*
        if insert tranx_node failed, print a warning message
        and turn off semi-sync
      */
      LogErr(WARNING_LEVEL, ER_SEMISYNC_FAILED_TO_INSERT_TRX_NODE,
             log_file_name, (ulong)log_file_pos);
      switch_off();
    }
  }
```

3. 同1, 当Master Rpl_semi_sync_master_status=ON 时， 连接上来的是异步复制，需要追上积压日志后会自动转为半同步复制?

如果当前发送事件的位置大于或等于“最大”提交事务binlog位置，则Slave现在已经赶上了，我们可以在这里切换半同步。 如果commit_file_name_inited_表示最近没有事务，我们可以立即启用半同步。

```
.\mysql-8.0.39\plugin\semisync\semisync_source.cc

int ReplSemiSyncMaster::try_switch_on(const char *log_file_name,
                                      my_off_t log_file_pos) {
  const char *kWho = "ReplSemiSyncMaster::try_switch_on";
  bool semi_sync_on = false;

  function_enter(kWho);

  /* If the current sending event's position is larger than or equal to the
   * 'largest' commit transaction binlog position, the slave is already
   * catching up now and we can switch semi-sync on here.
   * If commit_file_name_inited_ indicates there are no recent transactions,
   * we can enable semi-sync immediately.
   */
  if (commit_file_name_inited_) {
    int cmp = ActiveTranx::compare(log_file_name, log_file_pos,
                                   commit_file_name_, commit_file_pos_);
    semi_sync_on = (cmp >= 0);
  } else {
    semi_sync_on = true;
  }

  if (semi_sync_on) {
```

```
    /* Switch semi-sync replication on. */
    state_ = true;

    LogErr(INFORMATION_LEVEL, ER_SEMISYNC_RPL_SWITCHED_ON, log_file_name,
           (unsigned long)log_file_pos);
  }

  return function_exit(kWho, 0);
}
```

当Rpl_semi_sync_master_status=ON 时，更新数据包报头中的同步位，以向Slave指示Master是否会等待事件的回复。 如果半同步Rpl_semi_sync_master_status=OFF，我们检测到Slave赶上，我们就会打开半同步。

```
.\mysql-8.0.39\plugin\semisync\semisync_source.cc

int ReplSemiSyncMaster::updateSyncHeader(unsigned char *packet,
                                         const char *log_file_name,
                                         my_off_t log_file_pos,
                                         uint32 server_id) {

  ...

  if (is_on()) {
    /* semi-sync is ON */
    /* sync= false; No sync unless a transaction is involved. */

    if (reply_file_name_inited_) {
      cmp = ActiveTranx::compare(log_file_name, log_file_pos, reply_file_name_,
                                 reply_file_pos_);
      if (cmp <= 0) {
        /* If we have already got the reply for the event, then we do
         * not need to sync the transaction again.
         */
        goto l_end;
      }
    }

    if (wait_file_name_inited_) {
      cmp = ActiveTranx::compare(log_file_name, log_file_pos, wait_file_name_,
                                 wait_file_pos_);
    } else {
      cmp = 1;
    }

    /* If we are already waiting for some transaction replies which
     * are later in binlog, do not wait for this one event.
     */
    if (cmp >= 0) {
      /*
       * We only wait if the event is a transaction's ending event.
       */
```

```
      assert(active_tranxs_ != nullptr);
      sync = active_tranxs_->is_tranx_end_pos(log_file_name, log_file_pos);
    }
  } else {
    if (commit_file_name_inited_) {
      int cmp = ActiveTranx::compare(log_file_name, log_file_pos,
                                     commit_file_name_, commit_file_pos_);
      sync = (cmp >= 0);
    } else {
      sync = true;
    }
  }

  ...

l_end:
  unlock();

  /* We do not need to clear sync flag because we set it to 0 when we
   * reserve the packet header.
   */
  if (sync) {
    (packet)[2] = kPacketFlagSync;
  }

  return function_exit(kWho, 0);
}
```

读取slave返回的响应，根据响应头kPacketFlagSync是否已经有ack响应

```
.\mysql-8.0.39\plugin\semisync\semisync_source.cc

int ReplSemiSyncMaster::readSlaveReply(NET *net, const char *event_buf) {
  const char *kWho = "ReplSemiSyncMaster::readSlaveReply";
  int result = -1;

  function_enter(kWho);

  assert((unsigned char)event_buf[1] == kPacketMagicNum);
  if ((unsigned char)event_buf[2] != kPacketFlagSync) {
    /* current event does not require reply */
    result = 0;
    goto l_end;
  }

  /* We flush to make sure that the current event is sent to the network,
   * instead of being buffered in the TCP/IP stack.
   */
  if (net_flush(net)) {
    LogErr(ERROR_LEVEL, ER_SEMISYNC_SOURCE_FAILED_ON_NET_FLUSH);
    goto l_end;
  }
```

```
  net_clear(net, false);
  net->pkt_nr++;
  result = 0;
  rpl_semi_sync_source_net_wait_num++;

l_end:
  return function_exit(kWho, result);
}
```

5. Master根据什么信号判断连接过来的slave是采用半同步模式还是异步模式?

MySQL根据slave提供rpl_semi_sync_replica来判断是否为半同步模式的slave

```
.\mysql-8.0.39\plugin\semisync\semisync_source.cc

bool ReplSemiSyncMaster::is_semi_sync_slave() {
  int null_value;
  long long val = 0;
  get_user_var_int("rpl_semi_sync_replica", &val, &null_value);
  return val;
}
```