**Percona Toolkit for MySQL安装使用手册**

**一、percona-toolkit简介**

percona-toolkit是一组高级命令行工具的集合，用来执行各种通过手工执行非常复杂和麻烦的mysql和系统任务，这些任务包括：

* 检查master和slave数据的一致性
* 有效地对记录进行归档
* 查找重复的索引
* 对服务器信息进行汇总
* 分析来自日志和tcpdump的查询
* 当系统出问题的时候收集重要的系统信息

percona-toolkit源自Maatkit 和Aspersa工具，这两个工具是管理mysql的最有名的工具，现在Maatkit工具已经不维护了，请大家还是使用percona-toolkit吧！这些工具主要包括开发、性能、配置、监控、复制、系统、实用六大类，作为一个优秀的DBA，里面有的工具非常有用，如果能掌握并加以灵活应用，将能极大的提高工作效率。

**二、percona-toolkit工具包安装**

**1.  软件包下载**

访问<http://www.percona.com/software/percona-toolkit/>下载最新版本的Percona Toolkit

下载的RPM包为：percona-toolkit-2.2.14-1.noarch.rpm

**2.软件包安装**

安装软件的操作系统测试环境为Centos 6.5 64 位

安装方式用RPM包来进行安装，在安装之前需要检查Percona Toolkit所需的rpm包：

perl-IO-Socket-SSL-1.31-2.el6.noarch.rpm

perl-Net-LibIDN-0.12-3.el6.x86\_64.rpm

perl-Net-SSLeay-1.35-9.el6.x86\_64.rpm

perl-TermReadKey-2.30-13.el6.x86\_64.rpm

perl-Time-HiRes-1.9721-119.el6\_1.1.x86\_64.rpm

通过yum方式来进行安装，或者在Centos 6.5的ISO安装文件中查找此5个文件来进行安装。

注：如果安装环境缺少RPM包，则在安装时可能会报如下的错误：

[root@106-65 data]# rpm -ivh percona-toolkit-2.2.14-1.noarch.rpm

warning: percona-toolkit-2.2.14-1.noarch.rpm: Header V4 DSA signature: NOKEY, key ID cd2efd2a

error: Failed dependencies:

perl(DBD::mysql) >= 1.0 is needed by percona-toolkit-2.2.14-1.noarch

perl(Term::ReadKey) is needed by percona-toolkit-2.2.14-1.noarch

从上面的提示可以看出需要安装perl(DBD::mysql)和perl(Term::ReadKey)，在安装perl(DBD::mysql)时，也可能报如下的错误：

[root@106-65 data]# rpm -ivh perl-DBD-MySQL-4.014-1.el5.rfx.x86\_64.rpm

warning: perl-DBD-MySQL-4.014-1.el5.rfx.x86\_64.rpm: Header V3 DSA signature: NOKEY, key ID 6b8d79e6

error: Failed dependencies:

libmysqlclient.so.15()(64bit) is needed by perl-DBD-MySQL-4.014-1.el5.rfx.x86\_64

libmysqlclient.so.15(libmysqlclient\_15)(64bit) is needed by perl-DBD-MySQL-4.014-1.el5.rfx.x86\_64

此问题为需要安装msyql包（以mysql5.5 RPM包为例）中的MySQL-shared-compat-5.5.44-1.rhel5.x86\_64.rpm，安装完成后，perl-DBD-MySQL-4.014-1.el5.rfx.x86\_64.rpm即可安装成功。

Percona Toolkit安装所需的环境准备好后，既可将percona-toolkit-2.2.14-1.noarch.rpm上传到centos中进行安装：

[root@106-65 data]# rpm -ivh perl-DBD-MySQL-4.014-1.el5.rfx.x86\_64.rpm

warning: perl-DBD-MySQL-4.014-1.el5.rfx.x86\_64.rpm: Header V3 DSA signature: NOKEY, key ID 6b8d79e6

Preparing... ########################################### [100%]

1:perl-DBD-MySQL ########################################### [100%]

**三、percona-toolkit工具包使用**

Mysql主从复制数据的校验、一致性检查可以使用Percona Toolkit工具来进行验证。

主从数据的一致性校验是个头疼的问题，偶尔被业务投诉主从数据不一致，或者几个从库之间的数据不一致，这会令人沮丧。

通常我们仅有一种办法，热备主库，然后替换掉所有的从库。这不仅代价非常大，而且类似治标不治本的方案，让人十分不安。

因此我们需要合适的工具，至少帮我们回答下面三个问题：

* 从库延迟导致了用户看到的数据不一致，还是真的主从数据就不一致？
* 如果不一致，这个比例究竟多大？
* 下次还会出现吗？

回答清楚这几个问题，有助于我们决定是否修复，以及修复的方式，还可以帮我们找出不一致的数据，进而定位问题根源。而Percona Toolkit的pt-table-checksum正是我们想要的。

**（1）pt-table-checksum简介**

pt-table-checksum是著名的percona-toolkit工具集的工具之一。

它通过在主库执行基于statement的sql语句来生成主库数据块的checksum，把相同的sql语句传递到从库，并在从库上计算相同数据块的checksum，最后，比较主从库上相同数据块的checksum值，由此判断主从数据是否一致。

这种校验是分表进行的，在每个表内部又是分块进行的，而且pt工具本身提供了非常多的限流选项，因此对线上服务的冲击较小。

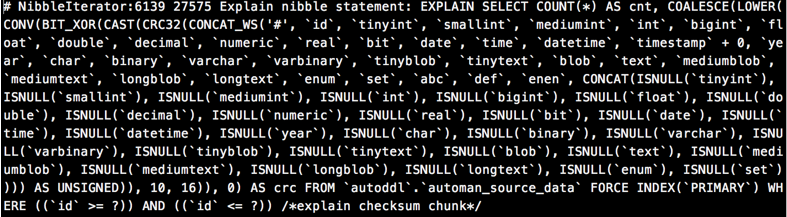
* **那么checksum这个值是如何计算出来的呢？**

1. 单行数据checksum值的计算

pt工具先检查表的结构，并获取每一列的数据类型，把所有数据类型都转化为字符串，然后用concat\_ws()函数进行连接，由此计算出该行的checksum值。checksum默认采用crc32，你可以自己定义效率更高的udf。

2. 数据块checksum值的计算

如果一行一行的计算checksum再去和从库比较，那么效率会非常低下。pt工具选择智能分析表上的索引，然后把表的数据split成一个个chunk，计算的时候也是以chunk为单位。因此引入了聚合函数BIT\_XOR()。它的功能可以理解为把这个chunk内的所有行的数据拼接起来，再计算crc32的值，就得到这个chunk的checksum值。sql语句如下：



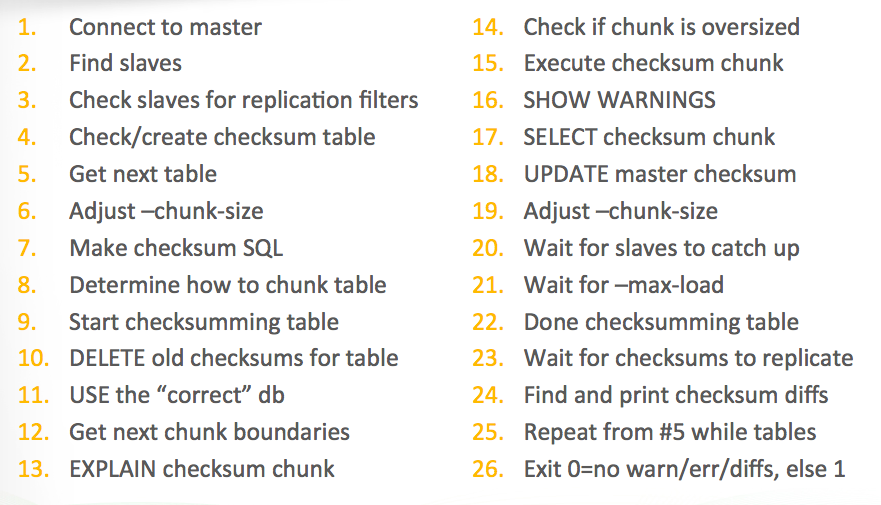
这其中还有count(\*)，用来计算chunk包含的行数。每一次对chunk进行checksum后，pt工具都会对耗时进行统计分析，并智能调整下一个chunk的大小，避免chunk太大对线上造成影响，同时也要避免chunk太小而效率低下。

3. 一致性如何保证

当pt工具在计算主库上某chunk的checksum时，主库可能还在更新，同时从库可能延迟使得relay-log中还有与这个chunk数据相关的更新，那该怎么保证主库与从库计算的是”同一份”数据？答案是加for update当前读锁，这保证了主库的某个chunk内部数据的一致性。否则，1000个人chekcusm同样的1000行数据，可能得到1000个不同的结果，你无法避开mvcc的干扰！获得for update锁后，pt工具开始计算chunk的checksum值，并把计算结果保存到pt工具自建的结果表中(采用replace into select的方式)，然后释放锁。该语句最终会传递到从库并执行相同的计算逻辑。

* **pt-table-checksum的内部工作过程**

有了上面关键的几点说明，我们再来看看pt工具的内部工作过程，如下图所示：



简单解释下工作过程：

1. 连接到主库：pt工具连接到主库，然后自动发现主库的所有从库。默认采用show full processlist来查找从库，但是这只有在主从实例端口相同的情况下才有效。

3. 查找主库或者从库是否有复制过滤规则：

这是为了安全而默认检查的选项。你可以关闭这个检查，但是这可能导致checksum的sql语句要么不会同步到从库，要么到了从库发现从库没有要被checksum的表，这都会导致从库同步卡库。

5. 开始获取表，一个个的计算。

6. 如果是表的第一个chunk，那么chunk-size一般为1000；如果不是表的第一个chunk，那么采用19步中分析出的结果。

7. 检查表结构，进行数据类型转换等，生成checksum的sql语句。

8. 根据表上的索引和数据的分布，选择最合适的split表的方法。

9. 开始checksum表。

10. 默认在chunk一个表之前，先删除上次这个表相关的计算结果。除非—resume。

14. 根据explain的结果，判断chunk的size是否超过了你定义的chunk-size的上限。如果超过了，为了不影响线上性能，这个chunk将被忽略。

15. 把要checksum的行加上for update锁，并计算。

17-18. 把计算结果存储到master\_crc master\_count列中。

19. 调整下一个chunk的大小。

20. 等待从库追上主库。如果没有延迟备份的从库在运行，最好检查所有的从库，如果发现延迟最大的从库延迟超过max-lag秒，pt工具在这里将暂停。

21. 如果发现主库的max-load超过某个阈值，pt工具在这里将暂停。

22. 继续下一个chunk，直到这个table被chunk完毕。

23-24. 等待从库执行完checksum，便于生成汇总的统计结果。每个表汇总并统计一次。

25-26. 循环每个表，直到结束。

校验结束后，在每个从库上，执行如下的sql语句即可看到是否有主从不一致发生：

select \* from percona.checksums where master\_cnt <> this\_cnt OR master\_crc <> this\_crc OR ISNULL(master\_crc) <> ISNULL(this\_crc) \G

* **缺陷和注意事项**

1、如果表没有主键或唯一索引，或者干脆没有任何索引，那么pt工具在chunk表的时候，将无所适从。不过我们已强制在建表的时候，每个表都必须有主键。

2、—check-binlog-format是默认选项，建议不要关闭它。pt-table-checksum工具自身产生的所有sql语句要基于语句格式同步到从库，这是由它的实现原理决定的。但是在A-B-C的级联复制结构中，如果B是行格式的复制，那么B与C的数据一致性校验就没法做了。在A上设置该sql语句为语句级并不会把set这个动作记录到binlog中，这个属性无法级联传递。

3、主从异构的情况下，checksum语句可能在从库上执行失败，即使是索引的不一致。例如sql语句中有force index某个索引，但是从库的表上没有这个索引，就会导致卡库。

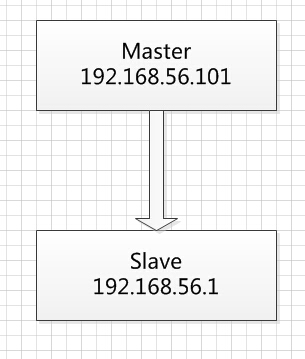
总结：

pt-table-checksum是校验主从数据不一致的最好工具。由于MySQL复制自身的缺陷，或主从切换不严谨，或备份软件bug等原因，都可能导致主从数据的不一致。不管你管不管，不一致都在那里，就看数据对你重不重要，重要的话，就定期做下检查吧。

* **pt-table-checksum示例操作，使用方法**

了解了pt-table-checksum运行原理后，通过示例来熟悉pt-table-checksum语法结构：

演示环境：

****

**在Master上初始化实验数据：**

create database mvbox;

use mvbox;

create table test(id int primary key,name varchar(20));

insert into test values(1,'a'),(2,'b'),(3,'c'),(4,'d');

**因为主从环境已经搭建，这些数据会自动同步到Slave上。**

**在Slave从库添加一个数据，模拟主从数据不一致的场景。**

insert into test values(5,'e');

**在Master主库执行pt-table-checksum命令：**

**pt-table-checksum --nocheck-replication-filters --no-check-slave-tables --set-vars innodb\_lock\_wait\_timeout=50 --databases=mvbox h=192.168.56.101,u=replication,p=replication,P=3306**

**执行以上命令后，会在192.168.56.101上创建percona库，在percona库中创建checksum表，用于存放所比较的记录。**

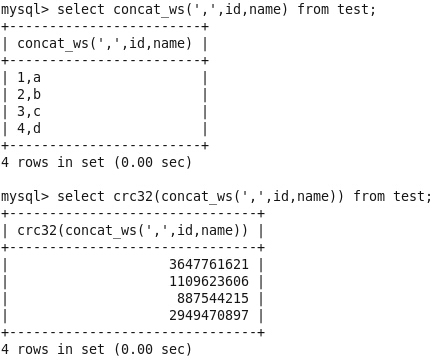
**所以执行pt-table-checksum命令的帐号,至少需要有全库的只读权限和percona库的读写权限**

create user **replication**;

GRANT SELECT, PROCESS, SUPER, REPLICATION SLAVE ON \*.\* TO ' **replication** '@'%' IDENTIFIED BY ' **replication** ';

grant all privileges on percona.\* TO ' **replication** '@'%' IDENTIFIED BY ' **replication** ';

**执行pt-table-checksum命令后，它会使用concat\_ws函数将数据合并为一行，然后使用crc32函数生成校验码,最后将其插入percona库的checksums表中。**

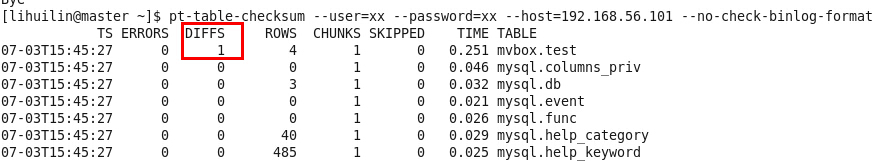
****

**因为主从环境，这个数据会复制到Slave，并在slave中创建percona库，在percona库中创建checksum表，用于存放所比较的记录。**

**也就是说Slave的percona.checksums表，存放的是主库数据的校验码。**

**所以在Slave对数据执行同样的校验,然后比对checksums表中的数据,就可以验证主从是否一致。**

**查看主从一致的情况：**

****

TS ：完成检查的时间。

ERRORS ：检查时候发生错误和警告的数量。

DIFFS ：0表示一致，1表示不一致。当指定--no-replicate-check时，会一直为0，当指定--replicate-check-only会显示不同的信息。

ROWS ：表的行数。

CHUNKS ：被划分到表中的块的数目。

SKIPPED ：由于错误或警告或过大，则跳过块的数目。

TIME ：执行的时间。

TABLE ：被检查的表名。

**pt-table-checksum命令常用参数：**

--nocheck-replication-filters ：不检查复制过滤器，建议启用。后面可以用--databases来指定需要检查的数据库。

--no-check-binlog-format : 不检查复制的binlog模式，要是binlog模式是ROW，则会报错。

--replicate-check-only :只显示不同步的信息。

--replicate= ：把checksum的信息写入到指定表中，建议直接写到被检查的数据库当中。

--databases= ：指定需要被检查的数据库，多个则用逗号隔开。

--tables= ：指定需要被检查的表，多个用逗号隔开

h=127.0.0.1 ：Master的地址

u=root ：用户名

p=123456：密码

P=3306 ：端口

更多的参数请见官网，上面指出来的是常用的，对该场景够用的参数。

**可以看到这个工具已经检测到了主从不一致的情况。  
如果发生不一致，可以使用pt-table-sync命令修复（后面会介绍）。**

**需要特别注意的是这个命令执行的过程中，会对表上共享锁，所以生产环境要慎重选择执行时间。**

好了，命令介绍完了，一起解释下上面执行的效果：  
通过**DIFFS** 是1 就可以看出主从的表数据不一致。怎么不一致呢？ 通过指定**--replicate=rep\_test.checksums**参数，就说明把检查信息都写到了checksums表中。默认会写入到**percona库的checksum表中。**  
进入SLAVE相应的库中查看checksums表的信息：

dba@192.168.56.1 : mvbox 05:19:13>select \* from checksums\G;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

db: mvbox

tbl: test

chunk: 1

chunk\_time: 0.000257

chunk\_index: NULL

lower\_boundary: NULL

upper\_boundary: NULL

this\_crc: b24c0933 #从的

this\_cnt: 3 #从的

master\_crc: f2890e1c #主的

master\_cnt: 4 #主的

ts: 2013-05-0817:18:181 row in set (0.00 sec)

通过上面的 this\_crc <> master\_crc 更能清楚的看出他们的不一致了，通过chunk知道是这个张表的哪个块上的记录出现不一致。要是主的binlog模式是Row 则会报错：

Replica db2 has binlog\_format ROW which could cause pt-table-checksum to break replication.

Please read "Replicas using row-based replication"in the LIMITATIONS section of the tool's documentation.

If you understand the risks, specify --no-check-binlog-format to disable this check.

从错误信息得出，要是不改binlog模式的话，则在执行上面的命令时候要指定：  
--no-check-binlog-format，**但是可能会破坏主从复制一致性，因此使用pt-table-checksum命令，主从复制binlog模式必须是在statement基于语句格式，而不是Row基于行模式。**

**（2）pt-table-sync简介**

从字面意思可以看出，它用来修复多个实例之间数据的不一致。它可以让主从的数据修复到最终一致，也可以使通过应用双写或多写的多个不相关的数据库实例修复到一致。

同时它还内部集成了pt-table-checksum的校验功能，可以一边校验一边修复，也可以基于pt-table-checksum的计算结果来进行修复。

* **工作原理**

1. 单行数据checksum值的计算

计算逻辑与pt-table-checksum一样，也是先检查表结构，并获取每一列的数据类型，把所有数据类型都转化为字符串，然后用concat\_ws()函数进行连接，由此计算出该行的checksum值。checksum默认采用crc32计算。

2. 数据块checksum值的计算

同pt-table-checksum工具一样，pt-table-sync会智能分析表上的索引，然后把表的数据split成若干个chunk，计算的时候以chunk为单位。可以理解为把chunk内所有行的数据拼接起来，再计算crc32的值，即得到该chunk的checksum值。

3. 坏块检测和修复

前面两步，pt-table-sync与pt-table-checksum的算法和原理一样。再往下，就开始有所不同：

pt-table-checksum只是校验，所以它把checksum结果存储到统计表，然后把执行过的sql语句记录到binlog中，任务就算完成。语句级的复制把计算逻辑传递到从库，并在从库执行相同的计算。

pt-table-checksum的算法本身并不在意从库的延迟，延迟多少都一样计算，不会影响计算结果的正确性(但是我们还是会检测延迟，因为延迟太多会影响业务，所以总是要加上—max-lag来限流)。

pt-table-sync则不同。它首先要完成chunk的checksum值的计算，一旦发现主从上同样的chunk的checksum值不同，就深入到该chunk内部，逐行比较并修复有问题的行。

其计算逻辑描述如下(以修复主从结构的数据不一致为例，业务双写的情况修复起来更复杂—因为涉及到冲突解决和基准选择的问题，限于篇幅，这里不介绍)：

1.对每一个从库，每一个表，循环进行如下校验和修复过程。

2.对每一个chunk，在校验时加上for update锁。一旦获得锁，就记录下当前主库的show master status值。

3.在从库上执行select master\_pos\_wait()函数，等待从库sql线程执行到show master status得到的位置。以此保证，主从上关于这个chunk的内容均不再改变。

4.对这个chunk执行checksum，然后与主库的checksum进行比较。

5.如果checksum相同，说明主从数据一致，就继续下一个chunk。

6.如果checksum不同，说明该chunk有不一致。深入chunk内部，逐行计算checksum并比较(单行的checksum的比较过程与chunk的比较过程一样，单行实际是chunk的size为1的特例)。

7.如果发现某行不一致，则标记下来。继续检测剩余行，直到这个chunk结束。

8.对找到的主从不一致的行，采用replace into语句，在主库执行一遍以生成该行全量的binlog，并同步到从库，这会以主库数据为基准来修复从库；对于主库有的行而从库没有的行，采用replace在主库上插入(必须不能是insert)；对于从库有而主库没有的行，通过在主库执行delete来删除(pt-table-sync强烈建议所有的数据修复都只在主库进行，而不建议直接修改从库数据；但是也有特例，后面会讲到)。

9.直到修复该chunk所有不一致的行。继续检查和修复下一个chunk。

10.直到这个从库上所有的表修复结束。开始修复下一个从库。

* **重要选项**

安全选项

—[no]check-triggers 检查是否有触发器，有则警告

—[no]foreign-key-checks 默认检查主外键约束，有则警告

—[no]unique-checks 检查是否有唯一索引，无则警告

过滤选项

—ignore-databases

—ignore-engines

—ignore-tables

其他选项

—replicate=s 与pt-table-checksum结合起来，只修复，而不校验。使用pt-table-checksum之前校验的结果

—bidirectional 双向同步。通常都以主库的数据为准，如果开启双向同步，就要定义冲突解决规则，会比较复杂

* **用法举例**

假设10.55.55.55是主库，10.73.73.73是它的从库，端口在3306。

1. 先校验：

./pt-table-checksum --user=user --password=pass --host=10.55.55.55 --port=3306 --databases=elink --tables=my\_cms\_10 --recursion-method=processlist

2. 根据校验结果，只修复10.73.73.73从库与主库不一致的地方：

./pt-table-sync --execute --replicate percona.checksums --sync-to-master h=10.73.73.73,P=3306,u=user,p=pass

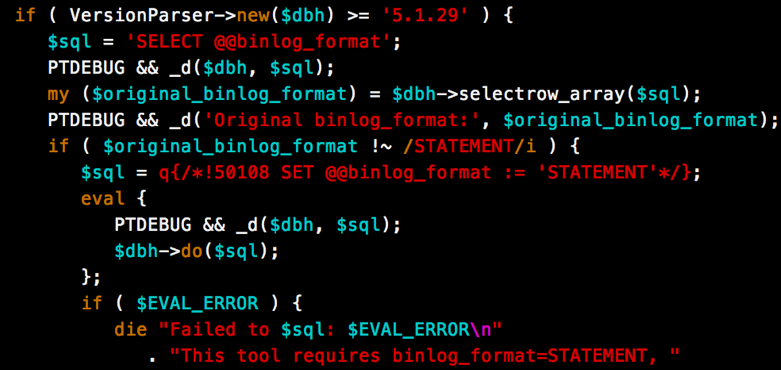
3. 修复后，再重新校验一次。执行第一步的语句即可。

4. 检查修复结果： 登陆到10.73.73.73，执行如下sql语句返回若为空，则说明修复成功： select \* from percona.checksums where master\_cnt <> this\_cnt OR master\_crc <> this\_crc OR ISNULL(master\_crc) <> ISNULL(this\_crc)

* **注意事项**

1、采用replace into来修复主从不一致，必须保证被replace的表上有主键或唯一键，否则replace into退化成insert into，起不到修复的效果。这种情况下pt-table-sync会采用其他校验和修复算法，但是效率非常低，例如对所有列的group by然后求count(\*)(表一定要有主键！)。

•主从数据不一致需要通过replace into来修复，该sql语句必须是语句级。pt-table-sync会把它发起的所有sql语句都设置为statement格式，而不管全局的binlog\_format值。这在级联A-B-C结构中，也会遇到pt-table-checksum曾经遇到的问题，引起行格式的中继库的从库卡库是必然。不过pt-table-sync默认会无限递归的对从库的binlog格式进行检查并警告：



2、由于pt-table-sync每次只能修复一个表，所以如果修复的是父表，则可能导致子表数据连带被修复，这可能会修复一个不一致而引入另一个不一致；如果表上有触发器，也可能遇到同样问题。所以在有触发器和主外键约束的情况下要慎用。pt-table-sync工具同样也不欢迎主从异构的结构。pt-table-sync工具默认会进行先决条件的检查。

3、pt-table-sync在修复过程中不能容忍从库延迟，这正好与pt-table-checksum相反。如果从库延迟太多，pt-table-sync会长期持有对chunk的for update锁，然后等待从库的master\_pos\_wait执行完毕或超时。从库延迟越大，等待过程就越长，主库加锁的时间就越长，对线上影响就越大。因此要严格设置max-lag。

4、对从库数据的修复通常是在主库执行sql来同步到从库。因此，在有多个从库时，修复某个从库的数据实际会把修复语句同步到所有从库。数据修复的代价取决于从库与主库不一致的程度，如果某从库数据与主库非常不一致.

举例说，这个从库只有表结构，那么需要把主库的所有数据重新灌一遍，然后通过binlog同步，同时会传递到所有从库。这会给线上带来很大压力，甚至拖垮集群。正确的做法是，先用pt-table-checksum校验一遍，确定不一致的程度：如果不同步的很少，用pt-table-sync直接修复；否则，用备份先替换它，然后用pt-table-sync修复。

说明： 这实际提供了一种对myisam备份的思路：如果仅有一个myisam的主库，要为其增加从库，则可以：先mysqldump出表结构到从库上，然后启动同步，然后用pt-table-sync来修复数据。

* **总结**

pt-table-sync工具很复杂也很强大。它修改数据库的内容，所以可能造成安全事故。

为了安全，建议：使用pt-table-checksum定期检测数据的一致性，并手动重建损坏较为严重的从库，最后才用pt-table-sync修复剩下的从库。