

MYSQL performance schema详解

0、performance_schema的介绍

MySQL的performance schema 用于监控MySQL server在一个较低级别的运行过程中的资源消耗、资源等待等情况。

特点如下：

1、提供了一种在数据库运行时实时检查server的内部执行情况的方法。performance_schema 数据库中的表使用performance_schema存储引擎。该数据库主要关注数据库运行过程中的性能相关的数据，与information_schema不同，information_schema主要关注server运行过程中的元数据信息

2、performance_schema通过监视server的事件来实现监视server内部运行情况，“事件”就是server内部活动中所做的任何事情以及对应的时间消耗，利用这些信息来判断server中的相关资源消耗在了哪里？一般来说，事件可以是函数调用、操作系统的等待、SQL语句执行的阶段（如sql语句执行过程中的parsing 或 sorting阶段）或者整个SQL语句与SQL语句集合。事件的采集可以方便的提供server中的相关存储引擎对磁盘文件、表I/O、表锁等资源的同步调用信息。

3、performance_schema中的事件与写入二进制日志中的事件（描述数据修改的events）、事件计划调度程序（这是一种存储程序）的事件不同。performance_schema中的事件记录的是server执行某些活动对某些资源的消耗、耗时、这些活动执行的次数等情况。

4、performance_schema中的事件只记录在本地server的performance_schema中，其下的这些表中数据发生变化时不会被写入binlog中，也不会通过复制机制被复制到其他server中。

5、当前活跃事件、历史事件和事件摘要相关的表中记录的信息。能提供某个事件的执行次数、使用时长。进而可用于分析某个特定线程、特定对象（如mutex或file）相关联的活动。

6、PERFORMANCE_SCHEMA存储引擎使用server源代码中的“检测点”来实现事件数据的收集。对于performance_schema实现机制本身的代码没有相关的单独线程来检测，这与其他功能（如复制或事件计划程序）不同

7、收集的事件数据存储于performance_schema数据库的表中。这些表可以使用SELECT语句查询，也可以使用SQL语句更新performance_schema数据库中的表记录（如动态修改performance_schema的setup_*开头的几个配置表，但要注意：配置表的更改会立即生效，这会影响到数据收集）

8、performance_schema的表中的数据不会持久化存储在磁盘中，而是保存在内存中，一旦服务器重启，这些数据会丢失（包括配置表在内的整个performance_schema下的所有数据）

9、MySQL支持的所有平台中事件监控功能都可用，但不同平台中用于统计事件时间开销的计时器类型可能会有所差异。

1、performance schema入门

在mysql的5.7版本中，性能模式是默认开启的，如果想要显式的关闭的话需要修改配置文件，不能直接进行修改，会报错Variable 'performance_schema' is a read only variable。

```
--查看performance_schema的属性
mysql> SHOW VARIABLES LIKE 'performance_schema';
+-----+-----+
| variable_name | value |
+-----+-----+
| performance_schema | ON |
+-----+-----+
1 row in set (0.01 sec)
```

--在配置文件中修改performance_schema的属性值，on表示开启，off表示关闭

```

[mysql]
performance_schema=ON

--切换数据库
use performance_schema;

--查看当前数据库下的所有表,会看到有很多表存储着相关的信息
show tables;

--可以通过show create table tablename来查看创建表的时候的表结构
mysql> show create table setup_consumers;
+-----+-----+
| Table          | Create Table
+-----+-----+
| setup_consumers | CREATE TABLE `setup_consumers` (
  `NAME` varchar(64) NOT NULL,
  `ENABLED` enum('YES','NO') NOT NULL
) ENGINE=PERFORMANCE_SCHEMA DEFAULT CHARSET=utf8 |
+-----+-----+
1 row in set (0.00 sec)

```

想要搞明白后续的内容，同学们需要理解两个基本概念：

instruments: 生产者，用于采集mysql中各种各样的操作产生的事件信息，对应配置表中的配置项我们可以称为监控采集配置项。

consumers: 消费者，对应的消费者表用于存储来自instruments采集的数据，对应配置表中的配置项我们可以称为消费存储配置项。

2、performance_schema表的分类

performance_schema库下的表可以按照监视不同的纬度就行分组。

```

--语句事件记录表，这些表记录了语句事件信息，当前语句事件表events_statements_current、历史语句事件表events_statements_history和长语句历史事件表events_statements_history_long、以及聚合后的摘要表summary，其中，summary表还可以根据帐号(account)，主机(host)，程序(program)，线程(thread)，用户(user)和全局(global)再进行细分
show tables like '%statement%';

--等待事件记录表，与语句事件类型的相关记录表类似：
show tables like '%wait%';

--阶段事件记录表，记录语句执行的阶段事件的表
show tables like '%stage%';

--事务事件记录表，记录事务相关的事件的表
show tables like '%transaction%';

--监控文件系统层调用的表
show tables like '%file%';

--监视内存使用的表
show tables like '%memory%';

--动态对performance_schema进行配置的配置表
show tables like '%setup%';

```

3、performance_schema的简单配置与使用

数据库刚刚初始化并启动时，并非所有instruments(事件采集项，在采集项的配置表中每一项都有一个开关字段，或为YES，或为NO)和consumers(与采集项类似，也有一个对应的事件类型保存表配置项，为YES就表示对应的表保存性能数据，为NO就表示对应的表不保存性能数据)都启用了，所以默认不会收集所有的事件，可能你需要检测的事件并没有打开，需要进行设置，可以使用如下两个语句打开对应的instruments和consumers (行计数可能会因MySQL版本而异)。

```
--打开等待事件的采集器配置项开关，需要修改setup_instruments配置表中对应的采集器配置项
UPDATE setup_instruments SET ENABLED = 'YES', TIMED = 'YES'where name like 'wait%';

--打开等待事件的保存表配置开关，修改setup_consumers配置表中对应的配置项
UPDATE setup_consumers SET ENABLED = 'YES'where name like '%wait%';

--当配置完成之后可以查看当前server正在做什么，可以通过查询events_waits_current表来得知，该表中每个线程只包含一行数据，用于显示每个线程的最新监视事件
select * from events_waits_current\G
***** 1. row *****
      THREAD_ID: 11
      EVENT_ID: 570
    END_EVENT_ID: 570
      EVENT_NAME: wait/synch/mutex/innodb/buf_dblwr_mutex
          SOURCE:
    TIMER_START: 4508505105239280
    TIMER_END: 4508505105270160
    TIMER_WAIT: 30880
          SPINS: NULL
    OBJECT_SCHEMA: NULL
    OBJECT_NAME: NULL
    INDEX_NAME: NULL
    OBJECT_TYPE: NULL
OBJECT_INSTANCE_BEGIN: 67918392
    NESTING_EVENT_ID: NULL
    NESTING_EVENT_TYPE: NULL
      OPERATION: lock
    NUMBER_OF_BYTES: NULL
          FLAGS: NULL
/*该信息表示线程id为11的线程正在等待buf_dblwr_mutex锁，等待事件为30880
属性说明:
    id:事件来自哪个线程，事件编号是多少
    event_name:表示检测到的具体的内容
    source:表示这个检测代码在哪个源文件中以及行号
    timer_start:表示该事件的开始时间
    timer_end:表示该事件的结束时间
    timer_wait:表示该事件总的花费时间
注意: _current表中每个线程只保留一条记录，一旦线程完成工作，该表中不会再记录该线程的事件信息
*/

/*
_history表中记录每个线程应该执行完成的事件信息，但每个线程的事件信息只会记录10条，再多就会被覆盖，*_history_long表中记录所有线程的事件信息，但总记录数量是10000，超过就会被覆盖掉
*/
select thread_id,event_id,event_name,timer_wait from events_waits_history order
by thread_id limit 21;

/*
```

summary表提供所有事件的汇总信息，该组中的表以不同的方式汇总事件数据（如：按用户，按主机，按线程等等）。例如：要查看哪些**instruments**占用最多的时间，可以通过对**events_waits_summary_global_by_event_name**表的**COUNT_STAR**或**SUM_TIMER_WAIT**列进行查询（这两列是对事件的记录数执行**COUNT(*)**、事件记录的**TIMER_WAIT**列执行**SUM(TIMER_WAIT)**统计而来）

```
*/
```

```
SELECT EVENT_NAME,COUNT_STAR FROM events_waits_summary_global_by_event_name  
ORDER BY COUNT_STAR DESC LIMIT 10;
```

```
/*
```

instance表记录了哪些类型的对象会被检测。这些对象在被**server**使用时，在该表中将会产生一条事件记录，例如，**file_instances**表列出了文件I/O操作及其关联文件名

```
*/
```

```
select * from file_instances limit 20;
```

4、常用配置项的参数说明

1、启动选项

performance_schema_consumer_events_statements_current=TRUE

是否在mysql **server**启动时就开启**events_statements_current**表的记录功能（该表记录当前的语句事件信息），启动之后也可以在**setup_consumers**表中**UPDATE**语句进行动态更新**setup_consumers**配置表中的**events_statements_current**配置项，默认值为**TRUE**

performance_schema_consumer_events_statements_history=TRUE

与**performance_schema_consumer_events_statements_current**选项类似，但该选项是用于配置是否记录语句事件短历史信息，默认为**TRUE**

performance_schema_consumer_events_stages_history_long=FALSE

与**performance_schema_consumer_events_statements_current**选项类似，但该选项是用于配置是否记录语句事件长历史信息，默认为**FALSE**

除了**statement**（语句）事件之外，还支持：**wait**（等待）事件、**state**（阶段）事件、**transaction**（事务）事件，他们与**statement**事件一样都有三个启动项分别进行配置，但这些等待事件默认未启用，如果需要在**MySQL Server**启动时一同启动，则通常需要写进**my.cnf**配置文件中

performance_schema_consumer_global_instrumentation=TRUE

是否在MySQL **Server**启动时就开启全局表（如：**mutex_instances**、**rwlock_instances**、**cond_instances**、**file_instances**、**users**、**hostsaccounts**、**socket_summary_by_event_name**、**file_summary_by_instance**等大部分的全局对象计数统计和事件汇总统计信息表）的记录功能，启动之后也可以在**setup_consumers**表中**UPDATE**语句进行动态更新全局配置项

默认值为**TRUE**

performance_schema_consumer_statements_digest=TRUE

是否在MySQL **Server**启动时就开启**events_statements_summary_by_digest**表的记录功能，启动之后也可以在**setup_consumers**表中**UPDATE**语句进行动态更新**digest**配置项

默认值为**TRUE**

performance_schema_consumer_thread_instrumentation=TRUE

是否在MySQL **Server**启动时就开启

events_xxx_summary_by_yyy_by_event_name表的记录功能，启动之后也可以在**setup_consumers**表中**UPDATE**语句进行动态更新线程配置项

默认值为**TRUE**

performance_schema_instrument[=name]

是否在MySQL **Server**启动时就启用某些采集器，由于**instruments**配置项多达数千个，所以该配置项支持**key-value**模式，还支持**%**号进行通配等，如下：

```
# [=name]可以指定为具体的Instruments名称（但是这样如果有多个需要指定的时候，就需要使用该选项多次），也可以使用通配符，可以指定instruments相同的前缀+通配符，也可以使用%代表所有的instruments
```

```
## 指定开启单个instruments
```

```
--performance-schema-instrument= 'instrument_name=value'
```

```
## 使用通配符指定开启多个instruments
```

```
--performance-schema-instrument= 'wait/synch/cond/%=COUNTED'
```

```
## 开关所有的instruments
```

```
--performance-schema-instrument= '%=ON'
```

```
--performance-schema-instrument= '%=OFF'
```

注意，这些启动选项要生效的前提是，需要设置**performance_schema=ON**。另外，这些启动选项虽然无法使用**show variables**语句查看，但我们可以通过**setup_instruments**和**setup_consumers**表查询这些选项指定的值。

2、系统变量

```
show variables like '%performance_schema%';
```

```
--重要的属性解释
```

```
performance_schema=ON
```

```
/*
```

控制**performance_schema**功能的开关，要使用MySQL的**performance_schema**，需要在**mysqld**启动时启用，以启用事件收集功能

该参数在5.7.x之前支持**performance_schema**的版本中默认关闭，5.7.x版本开始默认开启

注意：如果**mysqld**在初始化**performance_schema**时发现无法分配任何相关的内部缓冲区，则

performance_schema将自动禁用，并将**performance_schema**设置为OFF

```
*/
```

```
performance_schema_digests_size=10000
```

```
/*
```

控制**events_statements_summary_by_digest**表中的最大行数。如果产生的语句摘要信息超过此最大值，便无法继续存入该表，此时**performance_schema**会增加状态变量

```
*/
```

```
performance_schema_events_statements_history_long_size=10000
```

```
/*
```

控制**events_statements_history_long**表中的最大行数，该参数控制所有会话在

events_statements_history_long表中能够存放的总事件记录数，超过这个限制之后，最早的记录将被覆盖

全局变量，只读变量，整型值，5.6.3版本引入 * 5.6.x版本中，5.6.5及其之前的版本默认为10000，

5.6.6及其之后的版本默认值为-1，通常情况下，自动计算的值都是10000 * 5.7.x版本中，默认值为-1，

通常情况下，自动计算的值都是10000

```
*/
```

```
performance_schema_events_statements_history_size=10
```

```
/*
```

控制**events_statements_history**表中单个线程（会话）的最大行数，该参数控制单个会话在

events_statements_history表中能够存放的事件记录数，超过这个限制之后，单个会话最早的记录将被覆盖

全局变量，只读变量，整型值，5.6.3版本引入 * 5.6.x版本中，5.6.5及其之前的版本默认为10，5.6.6及其之后的版本默认值为-1，通常情况下，自动计算的值都是10 * 5.7.x版本中，默认值为-1，通常情况下，自动计算的值都是10

除了statement(语句)事件之外，wait(等待)事件、state(阶段)事件、transaction(事务)事件，他们与statement事件一样都有三个参数分别进行存储限制配置，有兴趣的同学自行研究，这里不再赘述

```
*/
performance_schema_max_digest_length=1024
/*
```

用于控制标准化形式的SQL语句文本在存入performance_schema时的限制长度，该变量与max_digest_length变量相关(max_digest_length变量含义请自行查阅相关资料)

全局变量，只读变量，默认值1024字节，整型值，取值范围0~1048576

```
*/
performance_schema_max_sql_text_length=1024
/*
```

控制存入events_statements_current，events_statements_history和events_statements_history_long语句事件表中的SQL_TEXT列的最大SQL长度字节数。超出系统变量performance_schema_max_sql_text_length的部分将被丢弃，不会记录，一般情况下不需要调整该参数，除非被截断的部分与其他SQL比起来有很大差异

全局变量，只读变量，整型值，默认值为1024字节，取值范围为0~1048576，5.7.6版本引入

降低系统变量performance_schema_max_sql_text_length值可以减少内存使用，但如果汇总的SQL中，被截断部分有较大差异，会导致没有办法再对这些有较大差异的SQL进行区分。增加该系统变量值会增加内存使用，但对于汇总SQL来讲可以更精准地区分不同的部分。

```
*/
```

5、重要配置表的相关说明

配置表之间存在相互关联关系，按照配置影响的先后顺序，可添加为

 image-20191203125003597

```
/*
performance_timers表中记录了server中有哪些可用的事件计时器
字段解释：
    timer_name:表示可用计时器名称，CYCLE是基于CPU周期计数器的定时器
    timer_frequency:表示每秒钟对应的计时器单位的数量，CYCLE计时器的换算值与CPU的频率相关、
    timer_resolution:计时器精度值，表示在每个计时器被调用时额外增加的值
    timer_overhead:表示在使用定时器获取事件时开销的最小周期值
*/
select * from performance_timers;

/*
setup_timers表中记录当前使用的事件计时器信息
字段解释：
    name:计时器类型，对应某个事件类别
    timer_name:计时器类型名称
*/
select * from setup_timers;

/*
setup_consumers表中列出了consumers可配置列表项
字段解释：
    NAME: consumers配置名称
    ENABLED: consumers是否启用，有效值为YES或NO，此列可以使用UPDATE语句修改。
*/
select * from setup_consumers;

/*
```


setup_instruments 表列出了**instruments** 列表配置项，即代表了哪些事件支持被收集：

字段解释：

NAME: **instruments**名称，**instruments**名称可能具有多个部分并形成层次结构

ENABLED: **instruments**是否启用，有效值为**YES**或**NO**，此列可以使用**UPDATE**语句修改。如果设置为**NO**，则这个**instruments**不会被执行，不会产生任何的事件信息

TIMED: **instruments**是否收集时间信息，有效值为**YES**或**NO**，此列可以使用**UPDATE**语句修改，如果设置为**NO**，则这个**instruments**不会收集时间信息

*/

```
SELECT * FROM setup_instruments;
```

/*

setup_actors表的初始内容是匹配任何用户和主机，因此对于所有前台线程，默认情况下启用监视和历史事件收集功能

字段解释：

HOST: 与**grant**语句类似的主机名，一个具体的字符串名字，或使用“%”表示“任何主机”

USER: 一个具体的字符串名称，或使用“%”表示“任何用户”

ROLE: 当前未使用，**MySQL 8.0**中才启用角色功能

ENABLED: 是否启用与**HOST**，**USER**，**ROLE**匹配的前台线程的监控功能，有效值为：**YES**或**NO**

HISTORY: 是否启用与**HOST**，**USER**，**ROLE**匹配的前台线程的历史事件记录功能，有效值为：**YES**或**NO**

*/

```
SELECT * FROM setup_actors;
```

/*

setup_objects表控制**performance_schema**是否监视特定对象。默认情况下，此表的最大行数为**100**行。

字段解释：

OBJECT_TYPE: **instruments**类型，有效值为：“**EVENT**”（事件调度器事件）、“**FUNCTION**”（存储函数）、“**PROCEDURE**”（存储过程）、“**TABLE**”（基表）、“**TRIGGER**”（触发器），**TABLE**对象类型的配置会影响表I/O事件（**wait/io/table/sql/handler instrument**）和表锁事件（**wait/lock/table/sql/handler instrument**）的收集

OBJECT_SCHEMA: 某个监视类型对象涵盖的数据库名称，一个字符串名称，或“%”（表示“任何数据库”）

OBJECT_NAME: 某个监视类型对象涵盖的表名，一个字符串名称，或“%”（表示“任何数据库内的对象”）

ENABLED: 是否开启对某个类型对象的监视功能，有效值为：**YES**或**NO**。此列可以修改

TIMED: 是否开启对某个类型对象的时间收集功能，有效值为：**YES**或**NO**，此列可以修改

*/

```
SELECT * FROM setup_objects;
```

/*

threads表对于每个**server**线程生成一行包含线程相关的信息，

字段解释：

THREAD_ID: 线程的唯一标识符（**ID**）

NAME: 与**server**中的线程检测代码相关联的名称（注意，这里不是**instruments**名称）

TYPE: 线程类型，有效值为：**BACKGROUND**、**BACKGROUND**。分别表示前台线程和后台线程

PROCESSLIST_ID: 对应**INFORMATION_SCHEMA.PROCESSLIST**表中的**ID**列。

PROCESSLIST_USER: 与前台线程相关联的用户名，对于后台线程为**NULL**。

PROCESSLIST_HOST: 与前台线程关联的客户端的主机名，对于后台线程为**NULL**。

PROCESSLIST_DB: 线程的默认数据库，如果没有，则为**NULL**。

PROCESSLIST_COMMAND: 对于前台线程，该值代表着当前客户端正在执行的**command**类型，如果是**sleep**则表示当前会话处于空闲状态

PROCESSLIST_TIME: 当前线程已处于当前线程状态的持续时间（秒）

PROCESSLIST_STATE: 表示线程正在做什么事情。

PROCESSLIST_INFO: 线程正在执行的语句，如果没有执行任何语句，则为**NULL**。

PARENT_THREAD_ID: 如果这个线程是一个子线程（由另一个线程生成），那么该字段显示其父线程**ID**

ROLE: 暂未使用

INSTRUMENTED: 线程执行的事件是否被检测。有效值：**YES**、**NO**

HISTORY: 是否记录线程的历史事件。有效值：**YES**、**NO** *

THREAD_OS_ID: 由操作系统层定义的线程或任务标识符（**ID**）：

*/

```
select * from threads
```

注意：在performance_schema库中还包含了很多其他的库和表，能对数据库的性能做完整的监控，大家需要参考官网详细了解。

6、performance_schema实践操作

基本了解了表的相关信息之后，可以通过这些表进行实际的查询操作来进行实际的分析。

--1、哪类的SQL执行最多？

```
SELECT DIGEST_TEXT,COUNT_STAR,FIRST_SEEN, LAST_SEEN FROM  
events_statements_summary_by_digest ORDER BY COUNT_STAR DESC
```

--2、哪类SQL的平均响应时间最多？

```
SELECT DIGEST_TEXT,AVG_TIMER_WAIT FROM events_statements_summary_by_digest ORDER  
BY COUNT_STAR DESC
```

--3、哪类SQL排序记录数最多？

```
SELECT DIGEST_TEXT,SUM_SORT_ROWS FROM events_statements_summary_by_digest ORDER  
BY COUNT_STAR DESC
```

--4、哪类SQL扫描记录数最多？

```
SELECT DIGEST_TEXT,SUM_ROWS_EXAMINED FROM events_statements_summary_by_digest  
ORDER BY COUNT_STAR DESC
```

--5、哪类SQL使用临时表最多？

```
SELECT DIGEST_TEXT,SUM_CREATED_TMP_TABLES,SUM_CREATED_TMP_DISK_TABLES FROM  
events_statements_summary_by_digest ORDER BY COUNT_STAR DESC
```

--6、哪类SQL返回结果集最多？

```
SELECT DIGEST_TEXT,SUM_ROWS_SENT FROM events_statements_summary_by_digest ORDER  
BY COUNT_STAR DESC
```

--7、哪个表物理IO最多？

```
SELECT file_name,event_name,SUM_NUMBER_OF_BYTES_READ,SUM_NUMBER_OF_BYTES_WRITE  
FROM file_summary_by_instance ORDER BY SUM_NUMBER_OF_BYTES_READ +  
SUM_NUMBER_OF_BYTES_WRITE DESC
```

--8、哪个表逻辑IO最多？

```
SELECT object_name,COUNT_READ,COUNT_WRITE,COUNT_FETCH,SUM_TIMER_WAIT FROM  
table_io_waits_summary_by_table ORDER BY sum_timer_wait DESC
```

--9、哪个索引访问最多？

```
SELECT OBJECT_NAME,INDEX_NAME,COUNT_FETCH,COUNT_INSERT,COUNT_UPDATE,COUNT_DELETE  
FROM table_io_waits_summary_by_index_usage ORDER BY SUM_TIMER_WAIT DESC
```

--10、哪个索引从来没有用过？

```
SELECT OBJECT_SCHEMA,OBJECT_NAME,INDEX_NAME FROM  
table_io_waits_summary_by_index_usage WHERE INDEX_NAME IS NOT NULL AND  
COUNT_STAR = 0 AND OBJECT_SCHEMA <> 'mysql' ORDER BY OBJECT_SCHEMA,OBJECT_NAME;
```

--11、哪个等待事件消耗时间最多？

```
SELECT EVENT_NAME,COUNT_STAR,SUM_TIMER_WAIT,AVG_TIMER_WAIT FROM  
events_waits_summary_global_by_event_name WHERE event_name != 'idle' ORDER BY  
SUM_TIMER_WAIT DESC
```

--12-1、剖析某条SQL的执行情况，包括statement信息，stage信息，wait信息

```
SELECT EVENT_ID,sql_text FROM events_statements_history WHERE sql_text LIKE  
'%count(*)%';
```

--12-2、查看每个阶段的时间消耗

```
SELECT event_id,EVENT_NAME,SOURCE,TIMER_END - TIMER_START FROM  
events_stages_history_long WHERE NESTING_EVENT_ID = 1553;
```

--12-3、查看每个阶段的锁等待情况

```
SELECT  
event_id,event_name,source,timer_wait,object_name,index_name,operation,nesting_e  
vent_id FROM events_waits_history_long WHERE nesting_event_id = 1553;
```


