

# MySQL 代码学习和调试

宋利兵

2012-7

- 一、获取 MySQL 源代码
- 二、编译 MySQL 源代码
- 三、运行 MySQL
- 四、调试 MySQL 源代码
- 五、阅读 MySQL 源代码

## 一、获取 MySQL 源代码



## 哪里有代码

- **Launchpad**

<https://launchpad.net/mysql>

<https://launchpad.net/maria>

- **MySQL Website**

<http://dev.mysql.com/downloads>

推荐从 *Launchpad* 获取代码，因为 *Launchpad* 上的代码是存储在代码库中的。包含了一些重要的信息，对理解代码很有帮助。

## 从 launchpad 取代码

### \* bzr 命令克隆

```
bzr branch lp:mysql-server/5.1
```

```
bzr branch lp:mysql-server/5.5
```

```
bzr branch lp:mysql-server/trunk
```

### \* 直接拷贝 bzr 库目录

.bzr 是 mysql 代码库中的隐藏目录

可以直接拷贝此目录来克隆 MySQL 代码库

### \* 更新代码

```
bzr pull
```

```
bzr pull lp:mysql-server/5.5
```

## 从 launchpad 取代码

### \* 创建公共代码库

为各个分支创建一个公共的代码库可以节约空间和克隆时间

```
mkdir bzrroot
```

```
bzr init-repo
```

```
cd bzrroot
```

```
bzr branch lp:mysql-server/trunk
```

```
bzr branch lp:mysql-server/5.5
```

```
Bzr branch lp:maria/5.5
```

## MySQL 的 Bazaar 代码库信息

```
-----
revno: 3820 [merge]
tags: clone-5.5.25-build, mysql-5.5.25
revision-id: annamalai.gurusami@oracle.com-20120510050316-6ct901cvswro4a5y
parent: georgi.kodinov@oracle.com-20120509140844-iit7r7vo2rtx6lm4
parent: annamalai.gurusami@oracle.com-20120510044831-d4xkpk2ky5sioeeq
committer: Annamalai Gurusami <annamalai.gurusami@oracle.com>
branch nick: mysql-5.5
timestamp: Thu 2012-05-10 10:33:16 +0530
message:
    Merging from mysql-5.1 to mysql-5.5.
-----
2 revno: 2661.810.8
   revision-id: annamalai.gurusami@oracle.com-20120510044831-d4xkpk2ky5sioeeq
   parent: sunanda.menon@oracle.com-20120508051914-6hjebro5b7tef2tx
   committer: Annamalai Gurusami <annamalai.gurusami@oracle.com>
   branch nick: mysql-5.1
   timestamp: Thu 2012-05-10 10:18:31 +0530
   message:
       Bug #14007649 65111: INNODB SOMETIMES FAILS TO UPDATE ROWS INSERTED
       BY A CONCURRENT TRANSACTION
3  The member function QUICK_RANGE_SELECT::init_ror_merged_scan() performs
   a table handler clone. Innodb does not provide a clone operation.
   The ha_innobase::clone() is not there. The handler::clone() does not
   take care of the ha_innobase->prebuilt->select lock type. Because of
```



## MySQL 的 Bazaar 代码库信息

### \* Revision Log

每次提交都会创建一个新的 revision. Revision log 中包含了丰富的信息。

- 版本信息    revno, revid, tags, parent
- 作者            committer
- 时间            timestamp
- 提交日志    message



## MySQL 的 Bazaar 代码库信息

### \* 提交日志

MySQL 的提交日志的格式

- Patch 的标题

Bug#ID <title>

Bug#<oracle bug id> <mysql bug id> <title>

*<http://bugs.mysql.com/>*

WL#ID <title>

*<http://forge.mysql.com/worklog/>*

- Bug 或 Feature 的描述

## MySQL 的 Bazaar 代码库信息

### - 显示 revision 日志

```
bzr log [--include-merges]
```

```
bzr log [--include-merges] -r <revision tag>
```

```
> bzr log -r li-bing.song@sun.com-20100328115733-41psnybf8hftnpug
```

```
-----  
revno: 2661.584.23
```

```
committer: <Li-Bing.Song@sun.com>
```

```
branch nick: mysql-5.1-bugteam
```

```
timestamp: Sun 2010-03-28 19:57:33 +0800
```

```
message:
```

```
Bug #50407 mysqlbinlog --database=X produces bad output for SAVEPOINTS
```



```
...
```

## MySQL 的 Bazaar 代码库信息

- 显示 revision 的 patch

```
bzr log [--include-merges] -r <revision> -p
```

```
> bzr log -p -r  
li-bing.song@sun.com-20100328115733-41psnybf8hftnpug
```

```
diff:    
=== modified file 'client/mysqlbinlog.cc'  
--- client/mysqlbinlog.cc      2010-02-17 18:07:28 +0000  
+++ client/mysqlbinlog.cc      2010-03-28 11:57:33 +0000  
@@ -730,9 +730,7 @@
```

## MySQL 的 Bazaar 代码库信息

- 显示每行代码的来源 (Revision)

```
bzr annotate <filename>
```

```
bzr annotate --show-ids <filename>
```

```
> bzr annotate --show-ids sql/rpl_rli.h
```

```
splr-lars@mysql.com-20051222053902-50313 | #include "rpl_tblmap.h"
```

```
splr-mats@kindahl-laptop.dnsalias.net-20070609051937-17444 | #include "rpl_reporting.h"
```

```
splr-cbell/Chuck@mysql_cab_desk.-20070727175837-21433 | #include "rpl_utility.h"
```

```
zhenxing.he@sun.com-20100622060300-16j08v7xxostkgpr | #include "log.h"
```

```
...
```

参考:<http://www.mysqlops.com/2011/11/16/bazaar-annotate-快速找到引入代码的版本.html>

## MySQL 的 Bazaar 代码库信息

### - Revision

#### \* Local Revision Number(revno)

It is a number, e.g. 1, 21, 1001, etc.

#### \* Global Revision ID(revid)

包含了作者，时间等信息。

```
'anders.song@greatopensource.com-20110115055451-dps4tyqxlbn7vgv'
```

当在分支之间进行合并时，revid 也会被合并进去。所以是全局唯一的。通常使用 revid 来查找一个 patch 的日志。

```
bzr help revisionspec
```

## Bazaar 的常用命令

<b>bzr commit</b>	提交代码到本地库中
<b>bzr uncommit</b>	从本地库中移除当前的 revision
<b>bzr revert</b>	将工作目录 (worktree) 代码返回到之前的某个版本
<b>bzr status</b>	查看工作目录的状态信息
<b>bzr push</b>	将本地的代码变更推送到其他的分支库中
<b>bzr merge</b>	合并分支直接的代码变更
<b>bzr help</b>	查看帮助信息
<b>bzr shelve</b>	将正在修改的代码放到一边 (架子上)
<b>bzr unshelve</b>	从架子上取回某个正在修改的代码





## 二、编译 MySQL 源代码



## MySQL 编译脚本

### \* MySQL 的编译脚本

MySQL 的代码中提供的一组编译脚本。开发调试时，使用这些脚本编译更简单快捷。这些脚本在 BUILD 目录中。

```
anders@anders-laptop:~/bzwwork/mysql-5.5$ cd BUILD/
anders@anders-laptop:~/bzwwork/mysql-5.5/BUILD$ ls
autorun.sh
build_mccge.sh
check-cpu
cleanup
cmake_configure.sh
compile-alpha
compile-alpha-debug
compile-amd64-debug-max
compile-amd64-debug-max-no-ndb
compile-amd64-gcov
compile-amd64-gprof
compile-amd64-max
compile-amd64-max-sci
compile-amd64-valgrind-max
compile-darwin-mwcc
compile-dist
compile-hpux11-parisc2-acc
compile-ia64-debug-max
compile-irix-mips64-mipspro
compile-ndb-autotest
compile-pentium
compile-pentium64
compile-pentium64-debug
compile-pentium64-debug-max
compile-pentium64-gcov
compile-pentium64-gprof
```

## 编译脚本分类

### \* 不同的平台

`compile-ppc-*`

`compile-pentium-*`

`compile-pentium64-*`

### \* 不同的功能

`compile-*-debug`      包含 MySQL 的调试代码

`compile-*-max`      包含所有的用户功能

`compile-*-valgrind` 包含 valgrind 调试功能

有的脚本同时包含机制功能如：

`compile-*-debug-max`

## 手动配置编译参数

- \* 最常用的是 `./BUILD/compile-pentium64-debug-max`

- \* `configure`

如果没有合适的方法，可以用 `configure` 配置后进行编译。

`BUILD/autorun.sh`

`./configure`

`make`

`sudo make install`

'configure' 命令有很多参数可以使用，可以使用下面的命令查询。

`configure --help`

## 编译 MySQL 代码

### \* 提示 \*

- 如果是第一次编译 MySQL，可能会碰到一些错误。多数是因为依赖的库和编译工具没有安装。根据错误提示进行安装就可以了。
- 这些脚本使用了 `-Werror` 等来进行严格的错误检查。因此用这些脚本可以尽早的发现 patch 中引入的代码问题。  
*对于写 patch 的同学，建议要使用编译脚本。*
- 这些脚本使用了 `-g` 参数，因此编译后代码可以很方便的进行调试。

### 三、运行 MySQL

## MySQL 的测试平台 mtr

### \* mtr

mysql\_code\_root/mysql-test/mtr

组成：perl 测试框架 + mysqltest + 测试用例

```
anders@anders-laptop:~/bzworwork/mysql-5.5/mysql-test$ ls
aaa.txt          include          mysql-test-run.pl  t
a.ann           lib             purify.sup        test.sh
CMakeFiles      Makefile        r                 test.sh~
cmake_install.cmake  Makefile.in    README            valgrind.sup
CMakeLists.txt   mtr            README.gcov       var
collections      mtr.out-of-source  README.stress
CTestTestfile.cmake  mysql-stress-test.pl  std_data
extra            mysql-test-run    suite
```

## mtr 的特点

### \* 简单

无需安装、无需配置、自动运行多实例

不改变系统环境、多版本的实例运行互不干扰

### \* 功能丰富

支持各种各样的 MySQL Server 配置

支持运行 MySQL 客户端程序和 shell 命令

支持 gdb, ddd, valgrind

压力测试 `mysql-stress-test.pl`

简单改造后，可以测试 mysql 的中间件



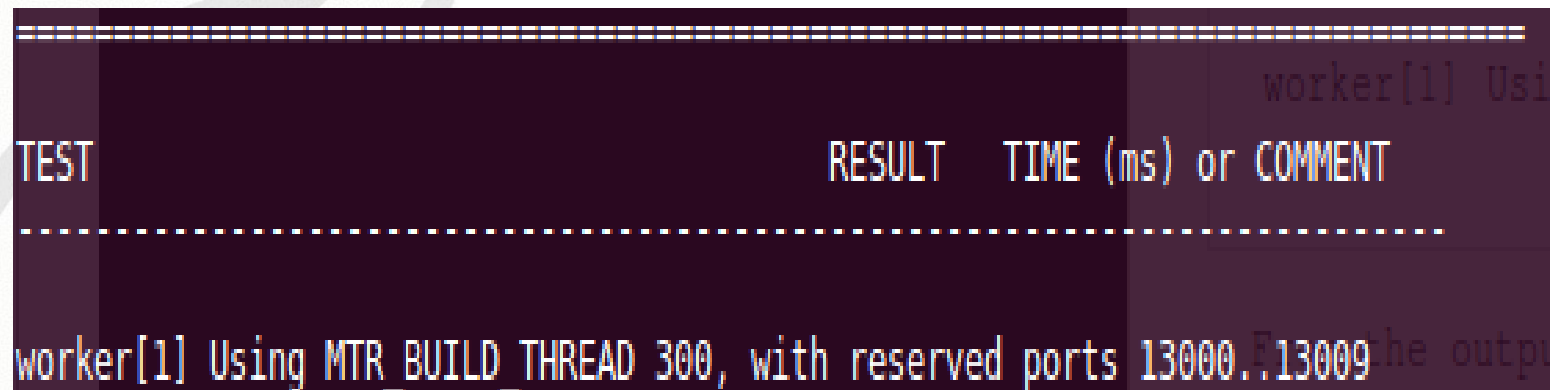
## 用 mtr 运行 MySQL Server

### \* 启动 MySQL 单实例

```
cd mysql-test
```

```
echo "sleep 1000000;" > t/my_run.test
```

```
./mtr my_run --testcase-timeout=1000000
```



TEST	RESULT	TIME (ms) or COMMENT
worker[1] Using MTR BUILD THREAD 300, with reserved ports 13000..13009		

使用的端口从 13000 开始，一个实例占一个端口。

## 用 mtr 运行 MySQL Server

### \* 启动 MySQL 双实例复制

```
cd mysql-test
```

```
echo "source include/master-slave.inc;" > suite/rpl/t/my_run.test
```

```
echo "sleep 1000000;" >> suite/rpl/t/my_run.test
```

```
./mtr my_run --testcase-timeout=1000000
```

Master -> mysqld.1    第一个端口

Slave    -> mysqld.2    第二个端口

## 设置 MySQL 的参数

\* mtr 参数 'mysqld'

```
./mtr my_run.test --mysqld='--innodb-file-per-table'
```

\* 为测试用例添加 option 文件

```
echo "--innodb-file-per-table" >t/my_run-master.opt
```

```
echo "--innodb-file-per-table" >t/my_run-slave.opt
```

\* 为测试用例添加 cnf 文件

```
t/my_run.cnf
```

启动多实例，就是在配置文件中配置多个 mysqld

## 连接到 MySQL Server

### \* TCP 连接

```
../client/mysql -u root --protocol=tcp -P13000
```

```
../client/mysql -u root --protocol=tcp -P13001
```

### \* Unix Domain Socket

mtr 的 socket 文件在 var/tmp 目录中

```
../client/mysql -u root -S var/tmp/mysqld.1.sock
```

```
../client/mysql -u root -S var/tmp/mysqld.2.sock
```

## MySQL 的数据和日志

### \* var 目录

`mysql_root/mysql_test/var`

数据和临时文件等所在目录

### \* 数据文件

数据 `var/mysql_d.*/data/`

### \* 日志文件

错误日志、调试日志 `var/log/`

General, Slow log `var/mysql_d.*/`

## 写测试用例

### \* SQL 语句

```
CREATE TABLE t1(c1 INT);  
--CREATE TABLE t1(c1 INT)  
delimiter /
```

### \* 引用其他的测试文件

```
--source include/have_binlog_format_row.inc  
--source extra/rpl_tests/rpl_innodb.test
```

### \* 切换 mysqltest 和 mysql server 的连接

```
--connection default  
--connection master  
--connection slave
```

## 写测试用例

### \* 复制同步

*--sync\_slave\_with\_master #on master connection*

*--save\_master\_pos # on master*

*--sync\_with\_master # on slave*

### \* 创建更多的连接

*--connect(conn\_name, localhost, root,,)*

*--connect(conn\_name,127.0.0.1,root,,test,\$SLAVE\_MYPORT)*

### \* 参考

mysqltest 手册

<https://dev.mysql.com/doc/mysqltest/2.0/en/index.html>

其他的测试用例 t/ suite/rpl suite/binlog suite/innodb



## 其他常用的功能

启用 binlog	<code>--source include/have_log_bin.inc</code>
启用 innodb	<code>--source include/have_innodb.inc</code>
使用指定的复制模式	<code>--source include/have_binlog_format_*.inc</code> * → statement, row, mixed
重启 Server	<code>--source include/restart_mysqld.inc</code> <code>--source include/rpl_start_server.inc</code> <code>--source include/rpl_stop_server.inc</code>
重启 Slave	<code>--source include/rpl_start_slaves.inc</code> <code>--source include/rpl_stop_slaves.inc</code>
检查主备数据的一致性	<code>--source include/rpl_diff.inc</code>
执行外部命令	<code>--exec \$MYSQL test -e "select * from t1\g"</code> <code>--exec \$MYSQL_DUMP --compact test t1</code> <code>--exec \$MYSQL_BINLOG --short-form ...</code> <code>--exec &lt;shell command&gt;</code>
期望返回特定的错误	<code>--error &lt;error_no&gt;</code>

## 四、调试 MySQL 源代码

## gdb 调试

\* gdb

`./mtr my_test --gdb`

\* ddd

`./mtr my_test -ddd`

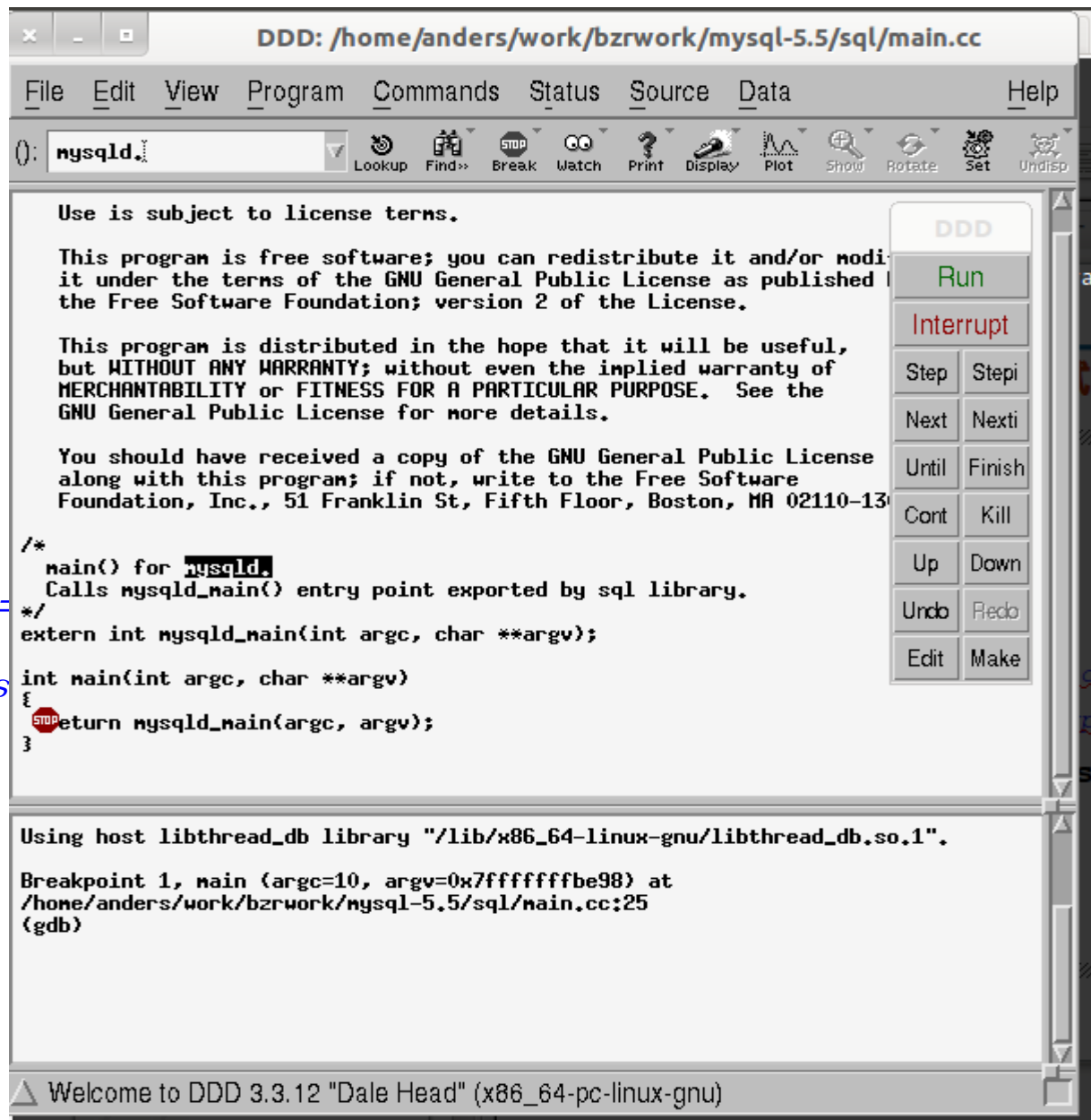
\* 单线程运行 MySQL

`./mtr my_test -ddd --mysqld=`

`'--thread-handling=no-threads'`

\* 禁用编译优化

优化后的执行代码和源代码不匹配，妨碍调试



## MySQL 的 debug 信息

### \* trace dump

```
./mtr my_test --debug
```

```
var/log/mysqld.*.trac
```

debug package 一个开源  
的代码追踪和调试工具

DEBUG\_ENTER,

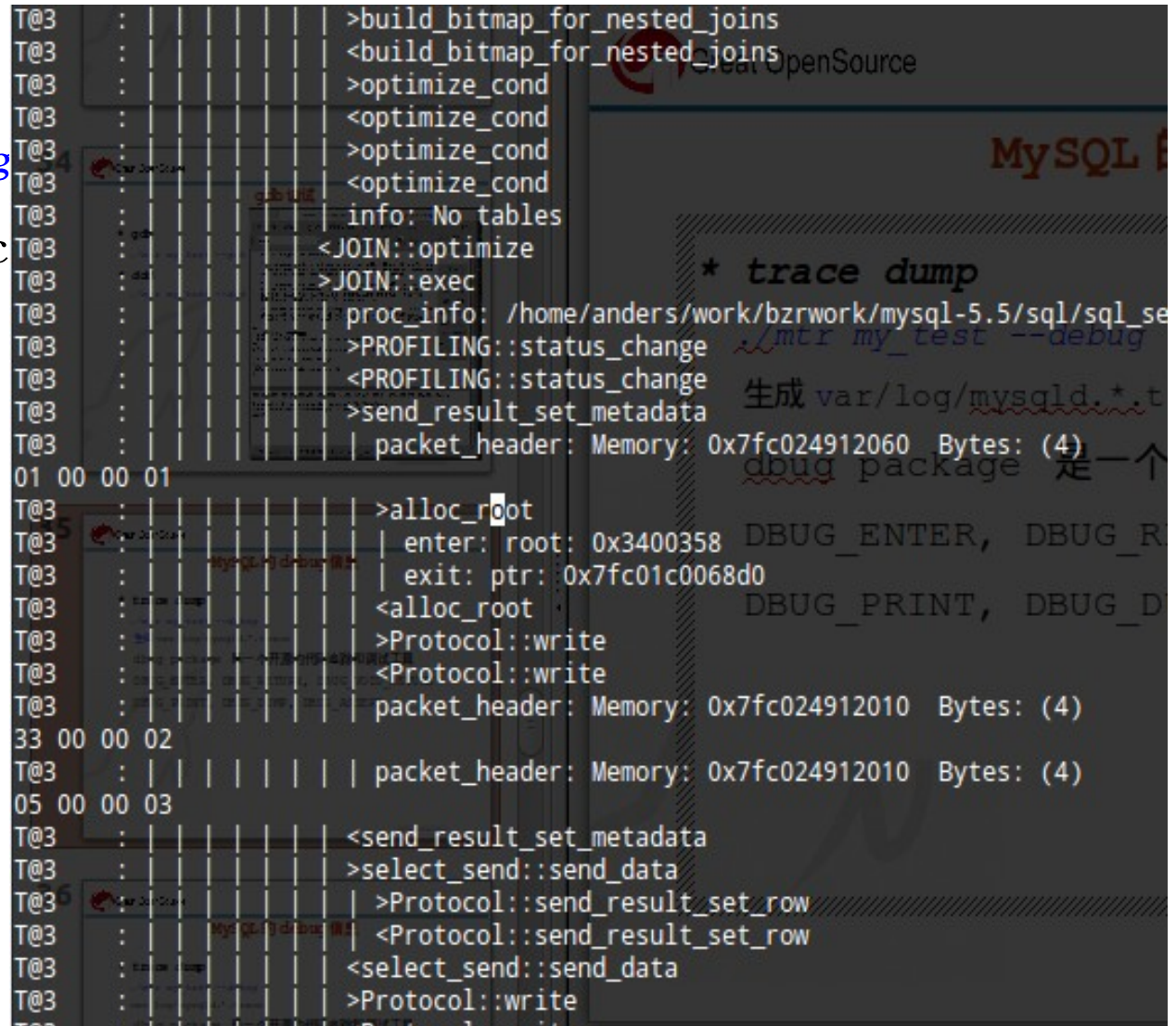
DEBUG\_RETURN,

DEBUG\_VOID\_RETURN

DEBUG\_PRINT,

DEBUG\_DUMP,

DEBUG\_ASSERT



```
T@3 : | | | | | | | | >build_bitmap_for_nested_joins
T@3 : | | | | | | | | <build_bitmap_for_nested_joins
T@3 : | | | | | | | | >optimize_cond
T@3 : | | | | | | | | <optimize_cond
T@3 : | | | | | | | | >optimize_cond
T@3 : | | | | | | | | <optimize_cond
T@3 : | | | | | | | | info: No tables
T@3 : | | | | | | | | <JOIN::optimize
T@3 : | | | | | | | | >JOIN::exec
T@3 : | | | | | | | | proc_info: /home/anders/work/bzrwork/mysql-5.5/sql/sql_se
T@3 : | | | | | | | | >PROFILING::status_change
T@3 : | | | | | | | | <PROFILING::status_change
T@3 : | | | | | | | | >send_result_set_metadata
T@3 : | | | | | | | | packet_header: Memory: 0x7fc024912060 Bytes: (4)
01 00 00 01
T@3 : | | | | | | | | >alloc_root
T@3 : | | | | | | | | enter: root: 0x3400358
T@3 : | | | | | | | | exit: ptr: 0x7fc01c0068d0
T@3 : | | | | | | | | <alloc_root
T@3 : | | | | | | | | >Protocol::write
T@3 : | | | | | | | | <Protocol::write
T@3 : | | | | | | | | packet_header: Memory: 0x7fc024912010 Bytes: (4)
33 00 00 02
T@3 : | | | | | | | | packet_header: Memory: 0x7fc024912010 Bytes: (4)
05 00 00 03
T@3 : | | | | | | | | <send_result_set_metadata
T@3 : | | | | | | | | >select_send::send_data
T@3 : | | | | | | | | >Protocol::send_result_set_row
T@3 : | | | | | | | | <Protocol::send_result_set_row
T@3 : | | | | | | | | <select_send::send_data
T@3 : | | | | | | | | >Protocol::write
T@3 : | | | | | | | | <Protocol::write
```

\* trace dump  
./mtr my\_test --debug  
生成 var/log/mysqld.\*.t  
debug package 是一个  
DEBUG\_ENTER, DEBUG\_R  
DEBUG\_PRINT, DEBUG\_D



## MySQL 的 debug point

### \* MySQL 调试宏

DEBUG\_EXECUTE

DEBUG\_EXECUTE\_IF

添加调试代码，通过 MySQL 变量控制调试代码何时执行

### \* 代码中添加调试点

```
DEBUG_EXECUTE_IF( "debug_point_name" , sleep(5););
```

### \* 启动执行调试点代码

```
SET SESSION debug=" d,debug_point_name" ;
```

### \* 提示： debug point 大多都有测试用例

代码中搜索 DEBUG\_EXECUTE\_IF

测试用例中搜索代码中定义的 <debug\_point\_name>

## MySQL 的 debug 信息

### \* 输出信息到 error 日志

`sql_print_error`

`sql_print_warning`

`sql_print_information`

### \* `strace-server`

*`./mtr my_test --strace-server`*

`var/log/mysqld.*.strace`

使用 `strace` 命令来跟踪 MySQL 的系统调用和 Signal

## 五、阅读 MySQL 源代码



## 主要的代码目录

目录	功能描述
sql	Server 的实现, 和 SQL 解析优化执行
storage	各种引擎代码
client	各种客户端工具代码
plugin	各种插件的代码
mysys	对系统调用的封装和一些共用算法
vio	对各种网络调用的封装
sql-common	server 和 client 共用的代码
libmysql	客户端 API
libmysqld	服务器端 API

最频繁查看的代码在 sql, storage/innobase, client

## 从哪里开始阅读代码

- \* `sql/mysql_d.cc`

`main()`

启动、配置文件、系统变量、网络、引擎、复制等各种功能初始化

- \* `sql/sql_parse.cc`

`dispatch_command()`

各种客户端命令的处理（包括 SQL），`COM_QUERY`，`COM_XXX`

`mysql_execute_command()`

各种 SQL 语句的执行，`SQLCOM_XXX`

从这两个文件开始，几乎总能找到你想找的代码。

## 从哪里开始阅读代码

- \* **storage/handler/ha\_innodb.cc**

InnoDB 对 MySQL 引擎 API 的实现。每个引擎都有自己的实现，主要代码的入口都在这里。

- \* **sql/log\_event.cc**

XXX\_log\_event::do\_apply\_event()

各种 binary log 在 slave 上的执行逻辑

- \* **sql/slave.cc**

handle\_slave\_io() IO 线程的处理逻辑

handle\_slave\_sql() SQL 线程的处理逻辑

## 从哪里开始阅读代码

\* `sql/sql_yacc.yy`

SQL 语句解析的代码



## 重要的数据结构

### \* THD(sql/sql\_class.h)

每个 session 有一个 THD, THD 中包含了这个线程特有的所有上下文信息。

变量名	描述
lex	解析后的 SQL 语句结构
query	当前连接的 SQL 语句
mem_root	线程的内存池，处理语句时不需要从系统中分配内存，从线程的内存池中分配。语句结束时释放回池中。
variables	SESSION 变量。THD 之外，另有 global_system_variables 存储 GLOBAL 变量。
host, user	当前连接的用户信息
transaction	事务相关的信息
...	

## 重要的数据结构

### \* TABLE(sql/sql\_table.h)

一个数据表可以有多个 TABLE 结构。每个线程在打开一个表时，会创建一个 TABLE，TABLE 不在线程间共享。

变量名	描述
s	TABLE_SHARE, 每个数据表只有一个 TABLE_SHARE, 在所有的 TABLE 之间共享。包含一些元数据信息。
file	引擎的 handler 实例, 和引擎的交互通过 file 来完成。
in_use	TABLE 所在的 THD
field	表的字段信息
record[2]	临时存储当前行的数据。在进行数据读写时, 引擎需要传入或传出行数据。select, insert, delete 只需要其中一个。Update 时需要 2 个, record[1] 指向修改前的数据, record[0] 指向修改后的数据。
read_set write_set	在引擎传入或传出一条记录时, 并不一定是完整的一行数据。当某些字段不需要时, 可以通过 read_set, write_set 告诉引擎。它们维护了一个字段是否需要(出现)的标志位。

## 重要的数据结构

### \* `SELECT_LEX(sql/sql_lex.h)`

解析后一个 SELECT 语句的结构信息。如果一个 SQL 中有多个 SELECT，需要用多个 `SELECT_LEX` 来表示。多层次的 `SELECT_LEX` 通过 `SELECT_LEX_UNIT` 来组织到一起。

变量名	描述
<code>table_list</code>	SELECT 中的用到的表，CREATE TABLE 时，也放在其中
<code>where, having</code>	WHERE 和 HAVING 中的内容
<code>group_list, order_list</code>	GROUP BY, ORDER BY 中的内容
<code>item_list</code>	SELECT 列表中的内容
<code>join, join_list</code>	Join 信息和 join 的表

*sql\_lex.h 中有详细的描述，搜索“The state of the lex parsing for selects”*



## 重要的数据结构

### \* LEX(sql/sql\_lex.h)

解析后的 SQL 语句信息，每个语句解析后生成 1 个 LEX 对象，记录在 THD::lex 中。

变量名	描述
unit select_lex current_select all_selects_lists	SELECT_LEX_UNIT, SELECT_LEX. 这 2 个结构的变量用来表达多层次的 SELECT 语句。Unit 是最外层的单元。select_lex 是语句中的第一个 select_lex。
create_info alter_info	CREATE TABLE, ALTER TABLE 的信息

*MySQL 中用 Item 代表 SQL 语句中的一个子单元。*

*Field、表达式、条件、函数、WHERE、HAVING、Limit 都是表达式*

## 阅读代码的小窍门

### \* 选择一个适合的阅读工具

*vim + ctags, cscope*

*emacs + cscope*

- 可以查找函数的定义和调用
- 可以查找符号（变量名，宏定义，结构体等）的定义和引用
- 可以列出文件中的函数、符号表
- 可以列出一个函数的函数调用列表

### \* 提示 \*

将 .yy 和 .ic 文件加入源码文件中

.yy -> sql\_yacc.yy 语法解析文件

.ic -> innodb 的 include 文件

## 阅读代码的小窍门

### \* 利用 bzd 来获取信息

patch message, [bugs.mysql.com](http://bugs.mysql.com),  
<https://forge.mysql.com/worklog>

### \* 通过错误信息查找代码

`sql/share/errmsg-utf8.txt` 包含所有错误信息

通过错误信息, 找到错误标签, 然后通过错误标签找到代码。

例: "PROCEDURE sp1 can't return a result set in the given context"

`errmsg-utf8.txt` → `ER_SP_BADSELECT`

`ER_SP_BADSELECT` → `sql_parse.cc::SQLCOM_CALL` 和 `MULTI_RESULT` 相关

## 阅读代码的小窍门

### \* 快读代码，跳过不相干的代码

- 注意宏定义，debug 的代码可以直接跳过。
- 注意宏定义，某些代码只属于特定的功能，如果和问题无关可以直接 跳过。如，

WITH\_PARTITION\_STORAGE\_ENGINE, HAVE\_REPLICATION

- 跨平台的代码可以忽略掉。MySQL 代码中对很多系统函数做了封装，功能没有任何变化，只是为了更好的跨平台。如：my\_malloc，my\_free，my\_write，my\_read 等。只需要理解其功能，不用关注实现细节。

## 阅读代码的小窍门

### \* 通过客户端程序的代码来查找 server 的代码

客户端程序一般都比较简单，更容易理解。在有些时候可以帮助更快的定位代码。

接上页的例子，MySQL 如何发送多结果集给客户端？

从 mysql 命令一定会处理这种情况。

mysql\_next\_result() -->

server\_status → SERVER\_MORE\_RESULTS\_EXISTS

mysql → mysql 通讯相关的代码

mysqlbinlog → binary log 相关的代码

## 阅读代码的小窍门

### \* 通过阅读 patch 来理解代码

<http://lists.mysql.com/commits>

可以选择一些自己关注的代码和功能的 patch 来阅读。比如想了解, MySQL 变量相关的代码, 找某个变量实现的 patch 来看。整个模块的代码就很清楚了。

对于希望从事 mysql server 代码开发的人, 可以选择自己关注的 bug 或 feature, 自己现实现一个, 然后和官方的 patch 对比。



## Q&A

谢谢！

