# 概述

参考：

<https://mp.weixin.qq.com/s/lBt5u3BiJueXsShJpxG5Yw>

<https://mp.weixin.qq.com/s/h12D0HS97Rjqeq8K1o5Lng>

<https://mp.weixin.qq.com/s/2JpPMauLq5aPkJTEMm6NuA>

<https://mp.weixin.qq.com/s/VCbJFFcJgPBDXdRTdFNtUg>

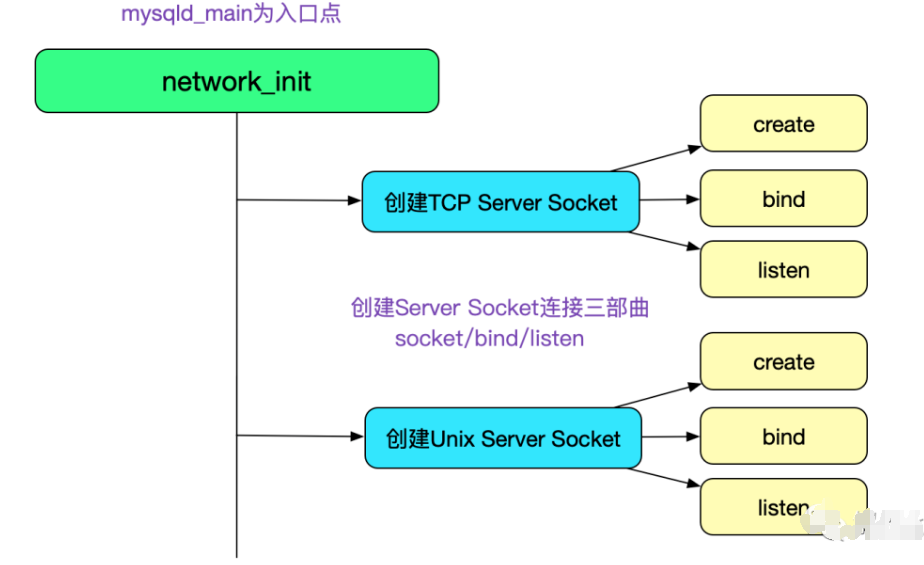
MySQL默认的连接控制方式采用的是每个连接使用一个**线程**执行客户端的请求。

MySQL的线程池是包含在企业版里面的服务器插件。使用线程池的目的是为了改善大量并发连接所带来的性能下降。在大量并发连接的工作负载下，使用线程池可以解决无法利用CPU缓存、上下文切换开销过大以及资源争用等问题。

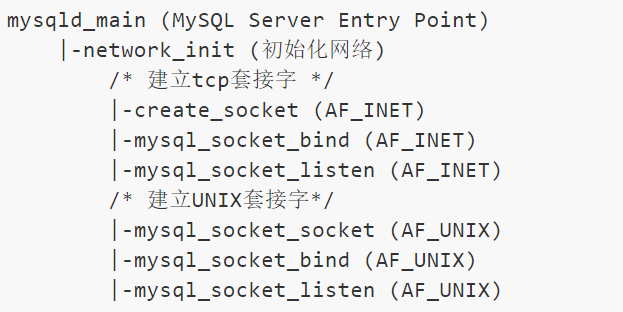
# 原理

## 启动socket监听

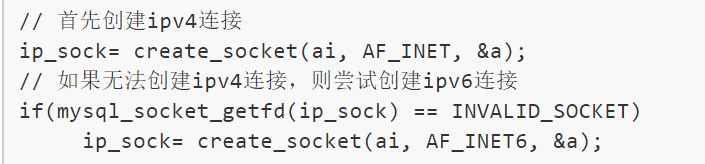
首先就需要找到其入口点，mysqld的入口点为mysqld\_main,跳过了各种配置文件的加载之后，来到了network\_init初始化网络环节，如下图所示：



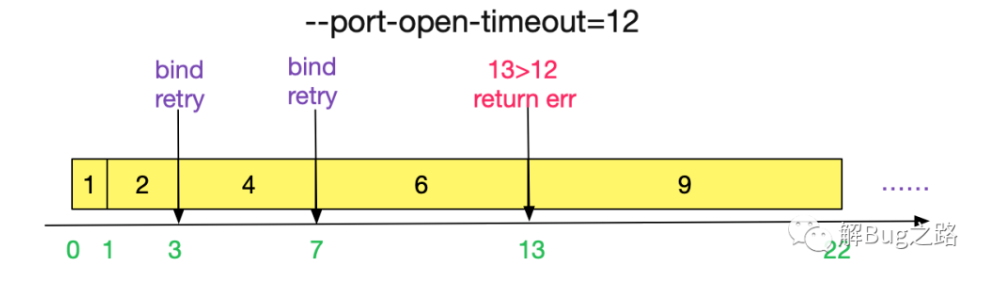
下面是其调用栈：



值得注意的是，在tcp socket的初始化过程中，考虑到了ipv4/v6的两种情况:

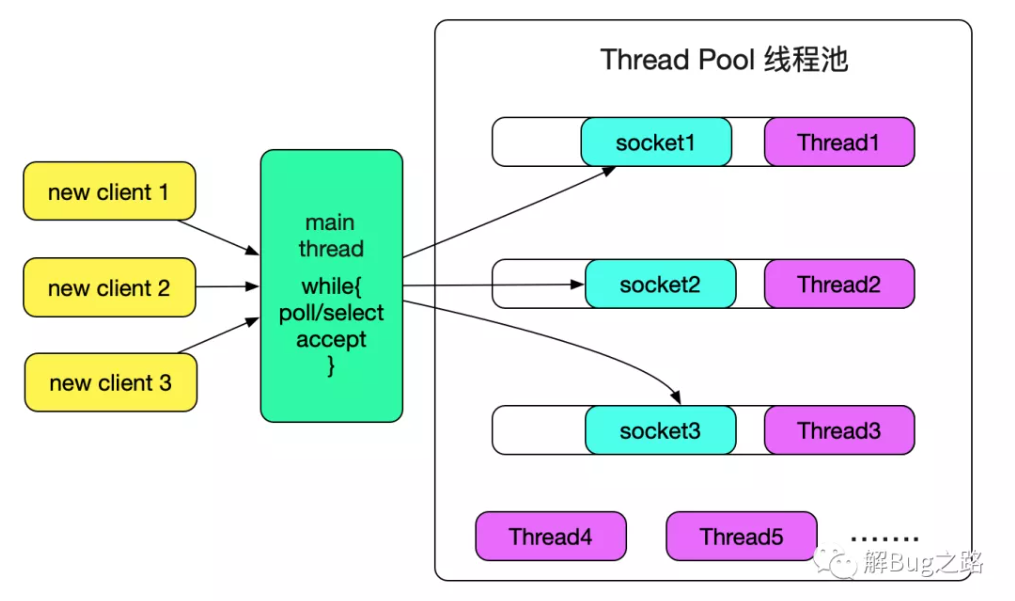


如果我们以很快的速度stop/start mysql，会出现上一个mysql的listen port没有被release导致无法当前mysql的socket无法bind的情况（这也是我们平时遇到的关闭后立即重启，然后连接mysql会报错稍等片刻可以连接成功的原因），在此种情况下mysql会循环等待，其每次等待时间为当前重试次数retry \* retry/3 +1秒，一直到设置的—port-open-timeout(默认为0)为止，如下图所示：



## 新建连接处理循环

通过handle\_connections\_sockets处理MySQL的新建连接循环，根据操作系统的配置通过poll/select处理循环(非epoll，这样可移植性较高，且mysql瓶颈不在网络上)。  
 MySQL通过线程池的模式处理连接(一个连接对应一个线程，连接关闭后将线程归还到池中)，如下图所示：

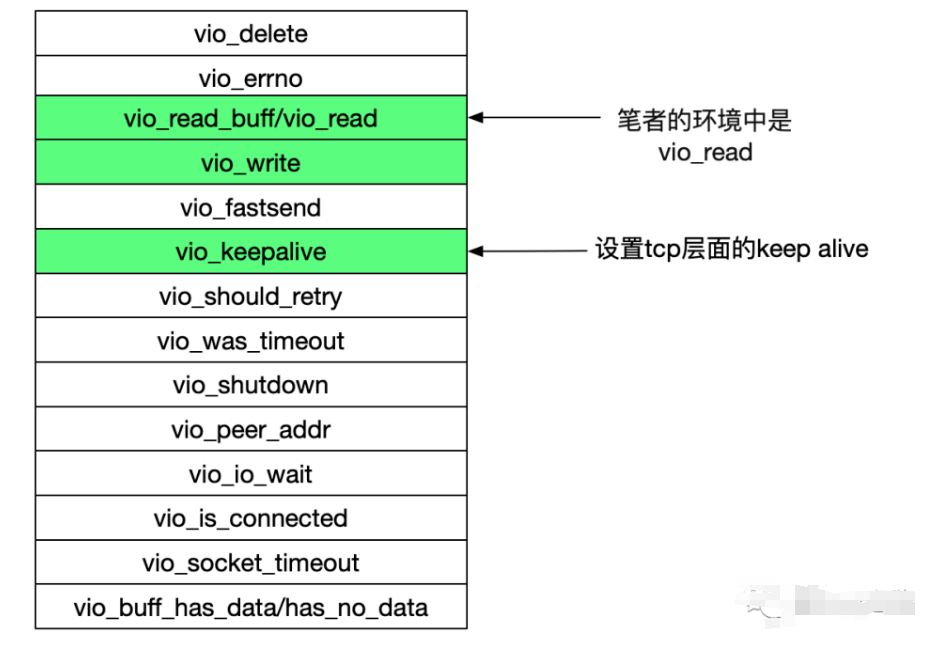


对应的调用栈如下所示：



## MySQL VIO

如上图代码中，每新建一个连接，都随之新建一个vio(mysql\_socket\_vio\_new->vio\_init)，在vio\_init的过程中，初始化了一堆回掉函数,如下图所示：

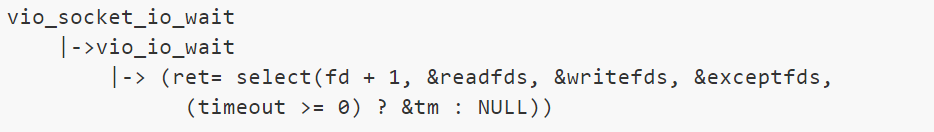


我们关注点在vio\_read和vio\_write上，如上面代码所示，在笔者所处机器的环境下将MySQL连接的socket设置成了非阻塞模式(O\_NONBLOCK)模式。所以在vio的代码里面采用了nonblock代码的编写模式，如下面源码所示：

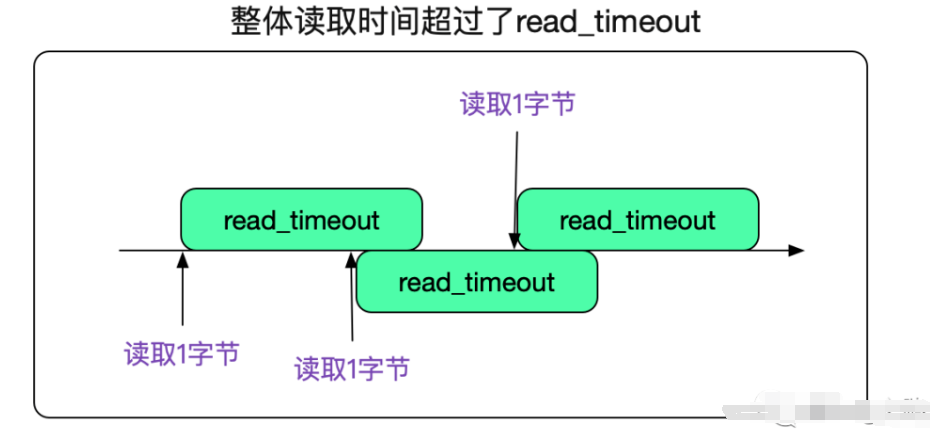
### vio\_read



即通过while循环去读取socket中的数据，如果读取为空，则通过vio\_socket\_io\_wait去等待(借助于select的超时机制)，其源码如下所示：



笔者在jdk源码中看到java的connection time out也是通过这,select(…wait\_time)的方式去实现连接超时的。  
 由上述源码可以看出，这个mysql的read\_timeout是针对每次socket recv(而不是整个packet的)，所以可能出现超过read\_timeout MySQL仍旧不会报错的情况，如下图所示：



### vio\_write

vio\_write实现模式和vio\_read一致，也是通过select来实现超时时间的判定,如下面源码所示：

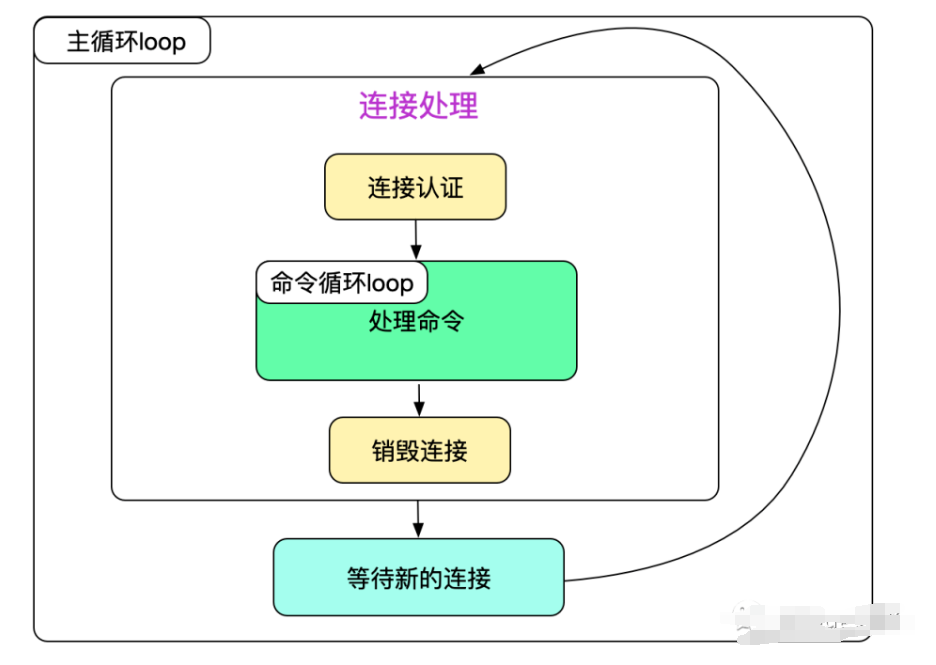


## 连接处理线程

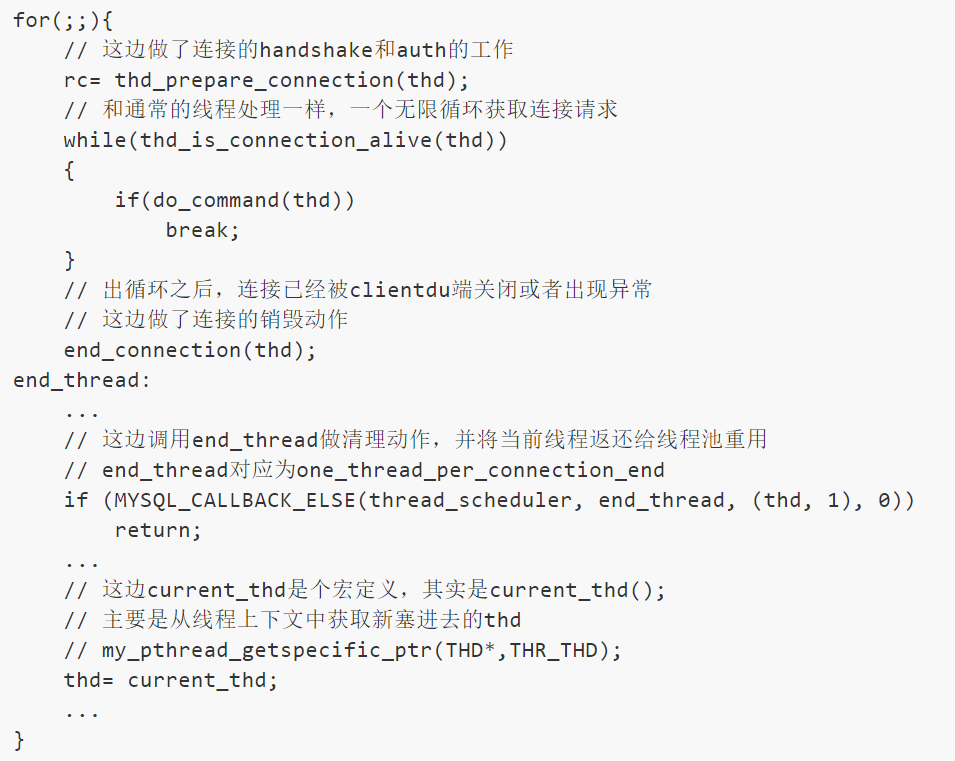
从上面的代码：



可以发现，MySQL每个线程的处理函数为handle\_one\_connection，其过程如下图所示：



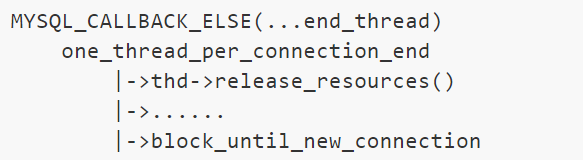
代码如下所示：



mysql的每个woker线程通过无限循环去处理请求。

## 线程的归还过程

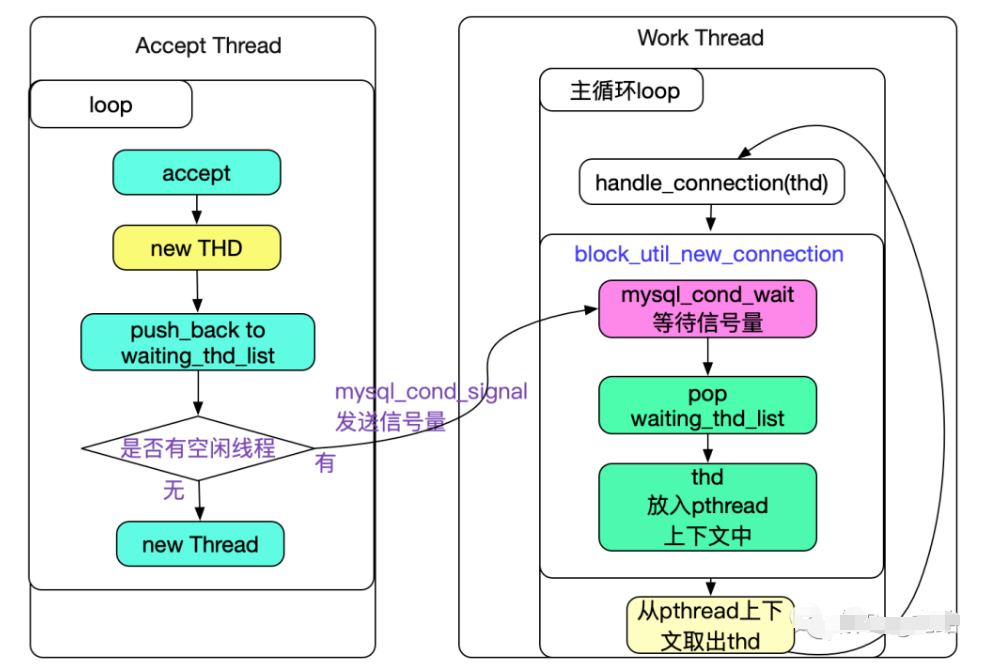
MySQL通过调用one\_thread\_per\_connection\_end(即上面的end\_thread)去归还连接。



线程在新连接尚未到来之前，等待在信号量上(下面代码是C/C++ mutex condition的标准使用模式)：



整个过程如下图所示：



由于MySQL的调用栈比较深，所以将thd放入线程上下文中能够有效的在调用栈中减少传递参数的数量。

## 总结

MySQL的网络IO模型采用了经典的线程池技术，虽然性能上不及reactor模型，但好在其瓶颈并不在网络IO上，采用这种方法无疑可以节省大量的精力去专注于处理sql等其它方面的优化。

# 参数

**thread\_pool\_size**

线程池是由一定数量的线程组（默认为16个通过thread\_pool\_size  
进行配置）构成，每个线程组管理一组客户端连接，最大连接数为4096。连接创建之后会以轮询的方式分配给线程组。连接池打破了每个连接与线程一一对应的关系，这一点与MySQL默认的线程控制方式不同，默认方式将一个线程与一个连接相关联，以便给定的线程从其连接执行所有的语句。

默认情况下，线程池试图确保每个组中每次最多执行一个线程，但有时为了获得最佳性能，允许临时执行多个线程。**每组里面有一个监听线程，负责监听分配给该组的连接**。线程会选择立即执行或稍后执行连接里面的语句，如果语句是唯一接收到的，并且当前没有排队或正在执行的语句，该语句就会立即执行。其它情况则会选择稍后执行。当该语句被判断为立即执行时，监听线程负责执行该语句，如果能够快速完成执行，该线程会返回监听状态，如果执行语句时间过长产生停滞，线程组会开启一个新的监听线程。线程池插件使用一个后台线程监控线程组状态，以确保线程组不会因为停滞的语句阻塞线程组。

## thread\_pool\_stall\_limit

可以通过thread\_pool\_stall\_limit 配置等待值时长，短等待值允许线程更快启动，也有助于避免死锁情况。长时间等待值对于长时间运行的工作负载非常有用，可以避免在当前语句执行时启动太多新语句。

## thread\_pool\_max\_active\_query\_threads

通过thread\_pool\_max\_active\_query\_threads设置运行的最大线程，如果该值不为0，则该数值为允许运行的最大线程数量，设置为0使用默认最大值。

线程池侧重于限制短时间运行语句的并发数量。在执行语句达到待值时长之前，它会阻止其他语句开始执行。如果语句执行超过了待值时长，允许其继续执行，但不再阻止其他语句启动。通过这种方式，线程池尝试确保每个线程组中永远不会有超过一个的短时间运行语句，但可能有多个长时间运行的语句。

如果遇到磁盘I/O操作或用户级锁(行锁或表锁)，语句就会被阻塞，将导致线程组无法使用。线程池的回调功能，可以确保线程池立即启动该组中的新线程来执行另一条语句。当一个被阻塞的线程返回时，线程池允许它立即重新启动。

线程池包含两个队列，高优先级队列和低优先级队列。当前正在执行的语句及该事务后续关联的语句将进入高优先级队列，其它语句进入低优先级队列。

此外，线程池重用活跃的线程，以更好地利用CPU缓存。这是一个对性能有很大影响的调整。

## max\_connections

理论上，可能出现的最大线程数是 max\_connections和thread\_pool\_size的总和。当所有连接都处于执行模式，并且每个组都创建了一个额外的线程来监听，可能会发生这种情况。