

零声学院：Darren QQ 326873713

课程：<https://ke.qq.com/course/420945?tuin=137bb271>

日期：2022年9月13日

重点内容：

- 为什么要使用连接池
- 连接池的设计原理
- 连接池扩展
- 连接池连接数量和线程池数量的关系

## 1 池化技术

---

池化技术能够减少资源对象的创建次数，提高程序的响应性能，特别是在高并发下这种提高更加明显。使用池化技术缓存的资源对象有如下共同特点：

1. 对象创建时间长；
2. 对象创建需要大量资源；
3. 对象创建后可被重复使用

像常见的线程池、内存池、连接池、对象池都具有以上的共同特点。

## 2 什么是数据库连接池

---

定义：数据库连接池（Connection pooling）是程序启动时建立足够的数据库连接，并将这些连接组成一个连接池，由程序动态地对池中的连接进行申请，使用，释放。

大白话：创建数据库连接是一个很耗时的操作，也容易对数据库造成安全隐患。所以，在程序初始化的时候，集中创建多个数据库连接，并把他们集中管理，供程序使用，可以保证较快的数据库读写速度，还更加安全可靠。

这里讲的数据库，不单只是指Mysql，也同样适用于Redis。

## 3 为什么使用数据库连接池

---

### 1. 资源复用

由于数据库连接得到复用，避免了频繁的创建、释放连接引起的性能开销，在减少系统消耗的基础上，另一方面也增进了系统运行环境的平稳性（减少内存碎片以及数据库临时进程/线程的数量）。

### 2. 更快的系统响应速度

数据库连接池在初始化过程中，往往已经创建了若干数据库连接置于池中备用。此时连接的初始化工作均已完成。对于业务请求处理而言，直接利用现有可用连接，避免了从数据库连接初始化和释放过程的开销，从而缩减了系统整体响应时间。

### 3. 统一的连接管理，避免数据库连接泄露

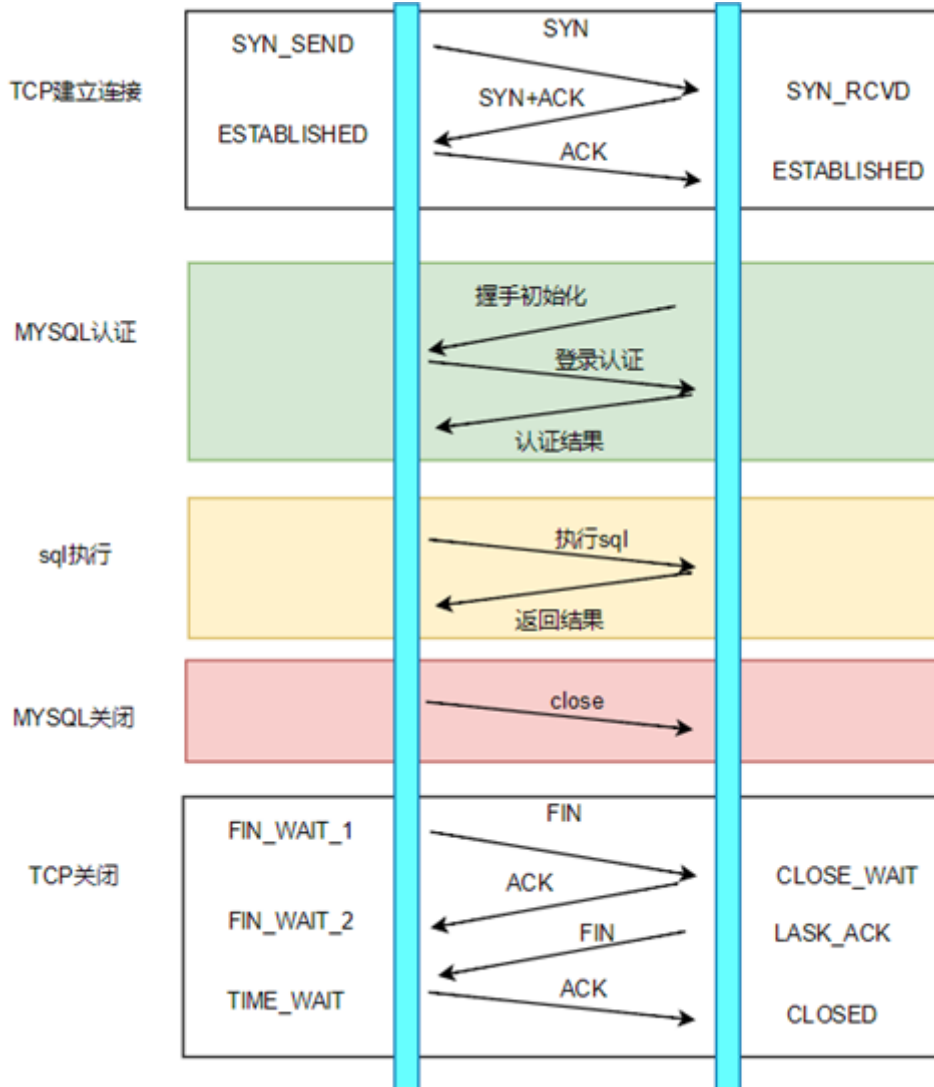
在较为完备的数据库连接池实现中，可根据预先的连接占用超时设定，强制收回被占用连接。从而避免了常规数据库连接操作中可能出现的资源泄露。

### 3.1 不使用连接池

---

1. TCP建立连接的三次握手（客户端与MySQL服务器的连接基于TCP协议）
2. MySQL认证的三次握手
3. 真正的SQL执行
4. MySQL的关闭
5. TCP的四次握手关闭

可以看到，为了执行一条SQL，需要进行TCP三次握手，Mysql认证、Mysql关闭、TCP四次挥手等其他操作，执行SQL操作在所有的操作占比非常低。



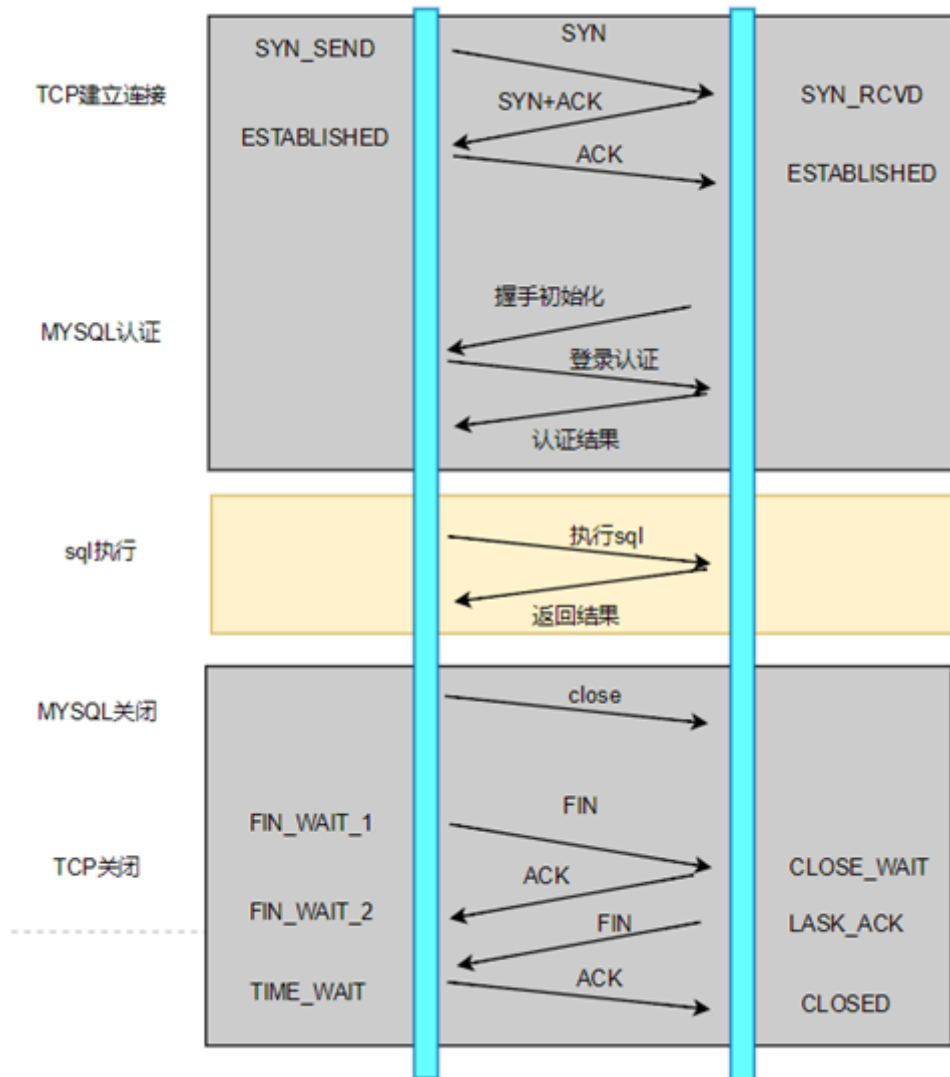
**优点\*\*：**实现简单 省了连接池的设计。 \*\*

**缺点：**每一次发起SQL操作都经历TCP建立连接、数据库用户身份验证、数据库用户登出、TCP断开连接。

- 网络IO较多
- 带宽利用率低
- QPS较低
- 应用频繁低创建连接和关闭连接，导致临时对象较多，带来更多的内存碎片
- 在关闭连接后，会出现大量TIME\_WAIT 的TCP状态（在2个MSL之后关闭）

## 3.2 使用连接池

第一次访问的时候，需要建立连接。但是之后的访问，均会复用之前创建的连接，直接执行SQL语句。



优点:

1. 降低了网络开销
2. 连接复用，有效减少连接数。
3. 提升性能，避免频繁的新建连接。新建连接的开销比较大
4. 没有TIME\_WAIT状态的问题

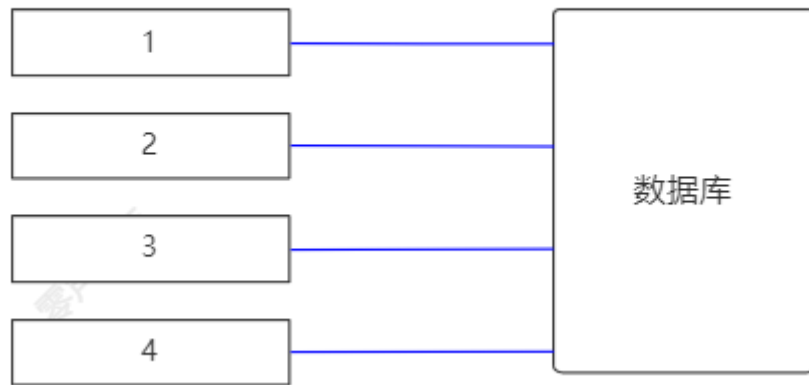
缺点:

1. 设计较为复杂

### 3.3 长连接和连接池的区别

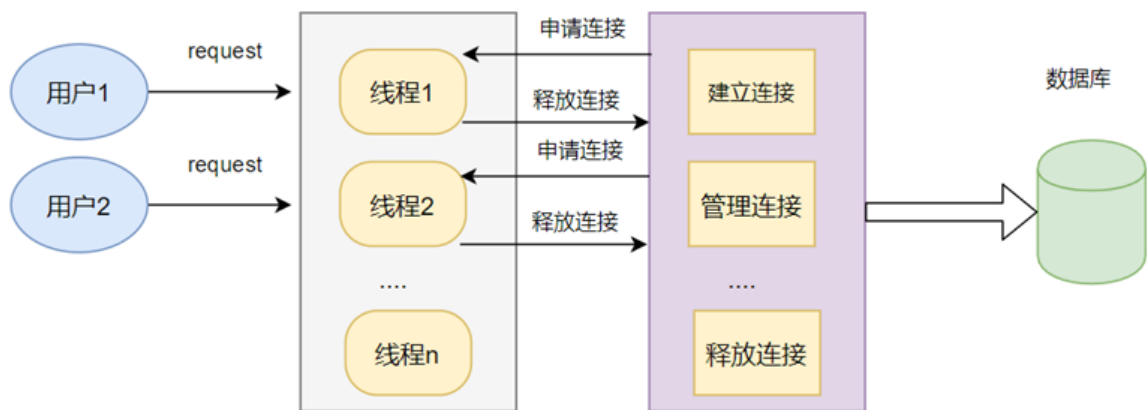
- 长连接是一些驱动、驱动框架、ORM工具的特性，由驱动来保持连接句柄的打开，以便后续的数据数据库操作可以重用连接，从而减少数据库的连接开销。
- 而连接池是应用服务器的组件，它可以通过参数来配置连接数、连接检测、连接的生命周期等。
- 连接池内的连接，其实就是长连接。

任务线程，比如4个线程      每个线程绑定一个连接

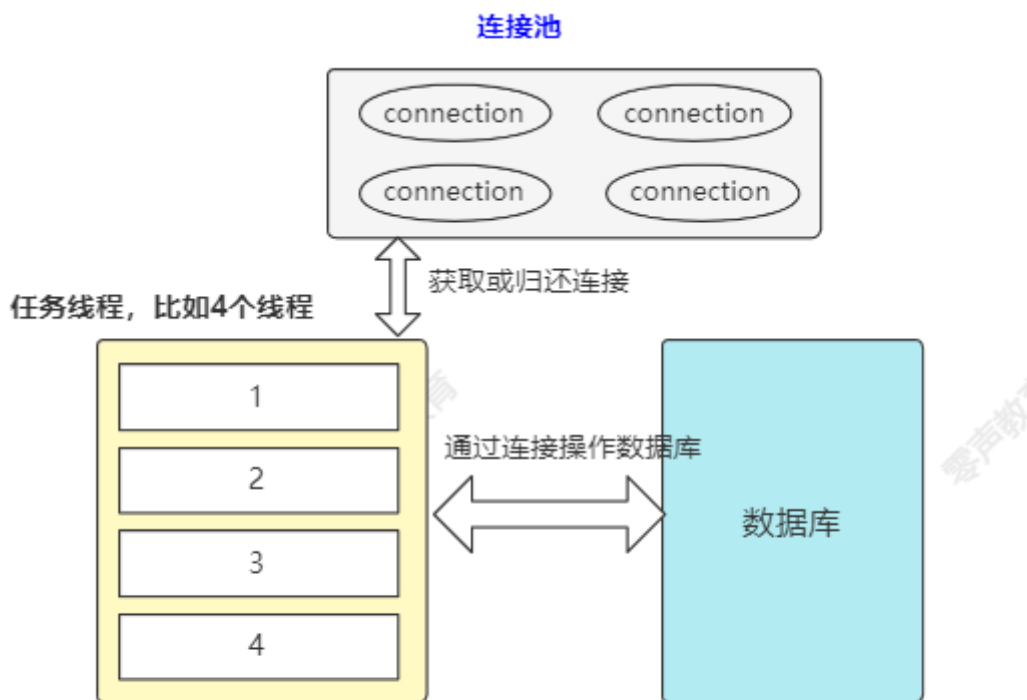


## 4 数据库连接池运行机制

1. 从连接池获取或创建可用连接；
2. 使用完毕之后，把连接返回给连接池；
3. 在系统关闭前，断开所有连接并释放连接占用的系统资源；



## 5 连接池和线程池的关系



**线程池：**主动操作，主动获取任务并执行任务。

**连接池：**被动操作（类似内存池），池的对象被任务获取，执行完任务后归还。

连接池和线程池的区别

- 线程池：主动调用任务。当任务队列不为空的时候从队列取任务取执行。
  - 比如去银行办理业务，窗口柜员是线程，多个窗口组成了线程池，柜员从排号队列叫号执行。
- 连接池：被动被任务使用。当某任务需要操作数据库时，只要从连接池中取出一个连接对象，当任务使用完该连接对象后，将该连接对象放回到连接池中。如果连接池中沒有连接对象可以用，那么该任务就必须等待。
  - 比如去银行用笔填单，笔是连接对象，我们要用笔的时候去取，用完了还回去。

连接池和线程池设置数量的关系

- 一般线程池线程数量和连接池连接对象数量一致；
- 一般线程执行任务完毕的时候归还连接对象；

## 6 线程池设计要点

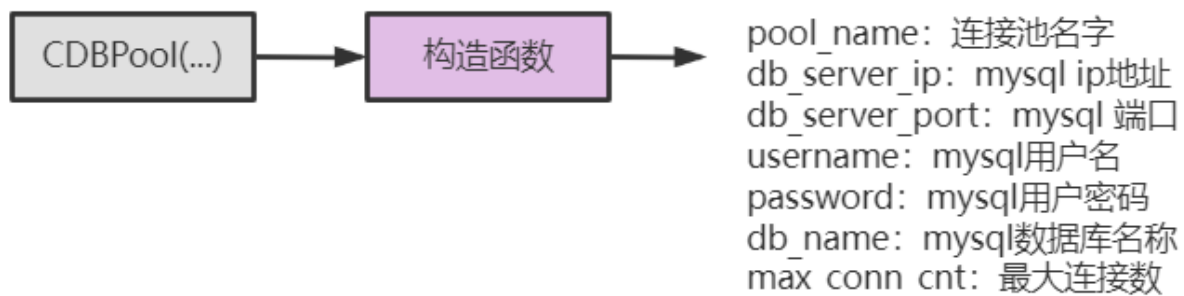
使用连接池需要预先建立数据库连接。

线程池设计思路：

1. 连接到数据库，涉及到数据库ip、端口、用户名、密码、数据库名字等；
  1. 连接的操作，每个连接对象都是独立的连接通道，它们是独立的
  2. 配置最小连接数和最大连接数
2. 需要一个队列管理他的连接，比如使用list；
3. 获取连接对象；
4. 归还连接对象；
5. 连接池的名字

## 6.1 连接池设计逻辑

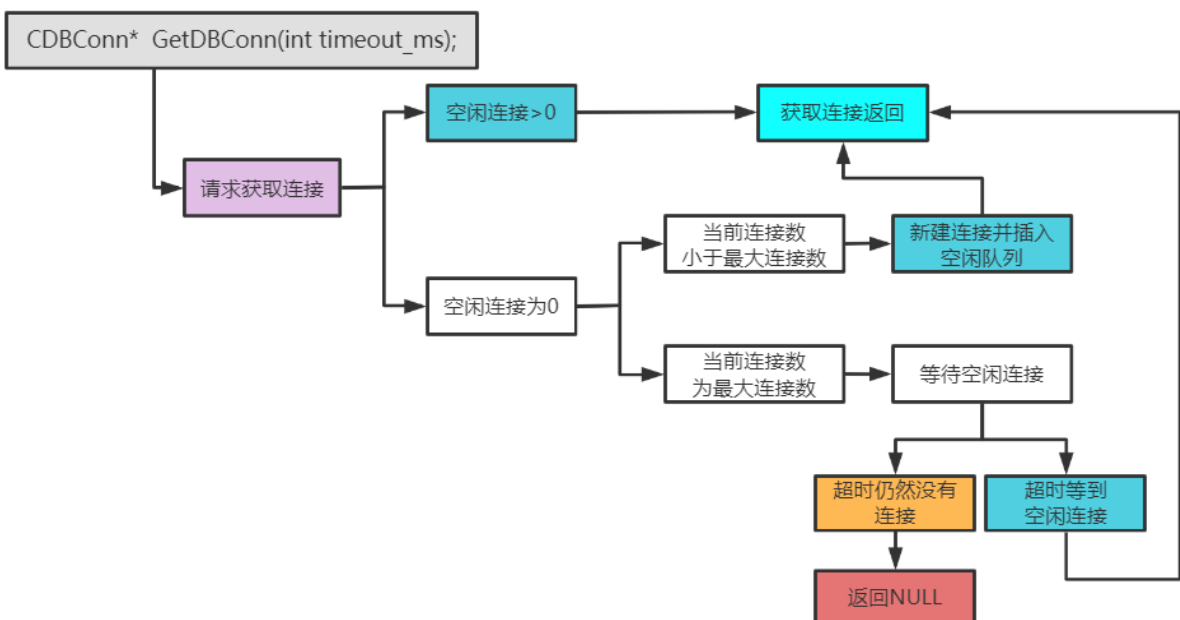
### 构造函数



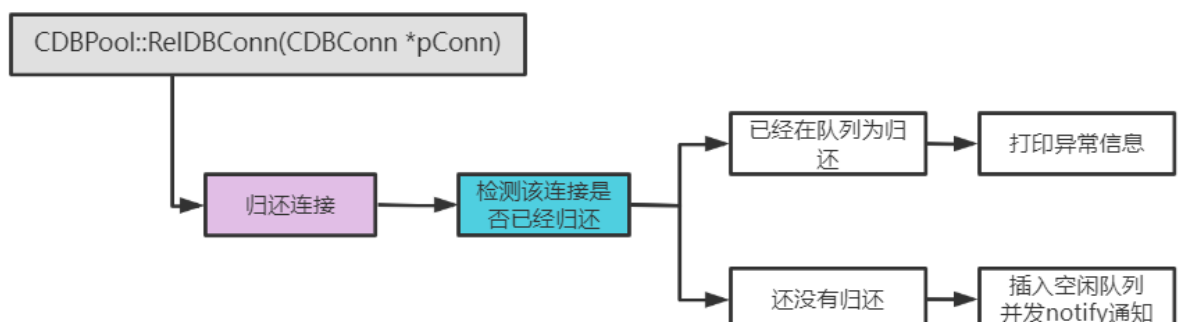
### 初始化



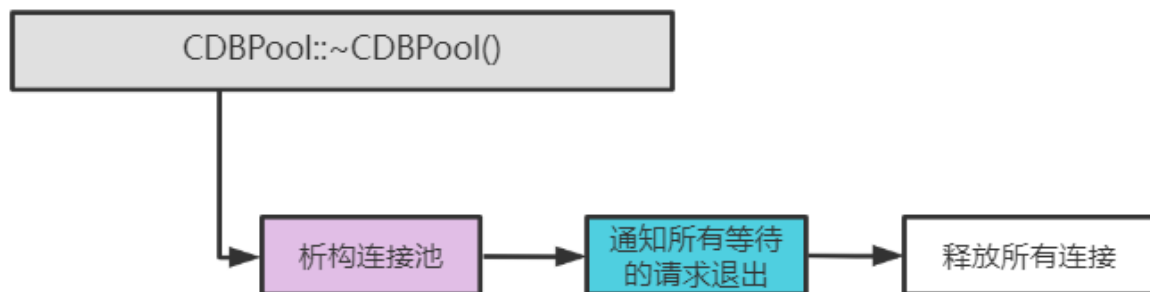
### 请求获取连接



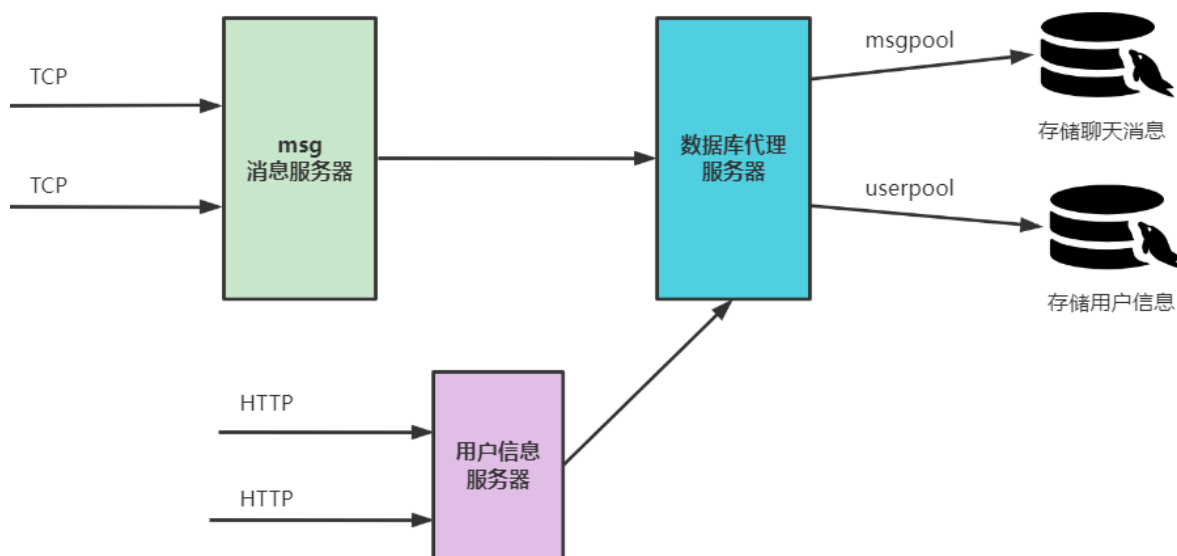
### 归还连接



### 析构连接池



## 连接池名



## 6.2 mysql连接重连机制

### 1. 设置启用（当发现连接断开时的）自动重连

```
my_bool reconnect = true;
```

```
mysql_options(m_mysql, MYSQL_OPT_RECONNECT, &reconnect); // 配合mysql_ping实现自动重连
```

### 2. 检测连接是否正常

```
int STDCALL mysql_ping(MYSQL *mysql);
```

描述：

检查与服务端的连接是否正常。**连接断开时，如果自动重新连接功能未被禁用，则尝试重新连接服务器。**该函数可被客户端用来检测闲置许久以后，与服务端的连接是否关闭，如有需要，则重新连接。

返回值：

连接正常，返回0；如有错误发生，则返回非0值。返回非0值并不意味着服务器本身关闭掉，也有可能是网络原因导致网络不通。

## 6.3 redis重连机制

1. 使用之前检测连接是否可用
2. 使用过程中出现连接异常则释放异常
3. 下一次使用该连接的时候如果发现连接不可用则重新初始化

redis的重连机制设计和mysql有区别。

✓ redis\_pool

```
int CacheConn::Init() // 重连探测
{
    if (m_pContext) // 非空，连接是正常的
    {
        // 1. redis 客户端接口编程的对象
        return 0;
    }

    // 1s 尝试重连一次
    uint64_t cur_time = (uint64_t)time(NULL);
    if (cur_time < m_last_connect_time + 1) // 重连尝试 间隔1秒
    {
        printf("cur_time:%lu, m_last_connect_time:%lu\n", cur_time, m_last_connect_time);
        return 1;
    }
    // printf("m_last_connect_time = cur_time\n");
    m_last_connect_time = cur_time;

    // 1000ms超时
    struct timeval timeout = {0, 1000000};
    // 重新连接
    // 建立连接后使用 redisContext 来保存连接状态。
    // redisContext 在每次操作后会修改其中的 err 和 errstr 字段来表示发生的错误码（大于0）和对应的描述。
    m_pContext = redisConnectWithTimeout(m_server_ip.c_str(), m_server_port, timeout);
}
```

## 7 连接池的具体实现

### 7.0 mysql常见命令

对数据库不熟悉的朋友参考：[MySQL 教程 | 菜鸟教程\(runoob.com\)](http://www.runoob.com/mysql/mysql-tutorial.html)

- CREATE DATABASE 数据库名; 创建数据库
- USE 数据库名; 选择数据库
- DROP DATABASE <数据库名>; 删除数据库
- CREATE TABLE table\_name (column\_name column\_type); 创建表
- DROP TABLE table\_name; 删除表
- SELECT column\_name,column\_name 查询  
FROM table\_name  
[WHERE Clause]  
[LIMIT N][ OFFSET M]

查询实例读取数据表：

```
select * from runoob_tbl;
```

### 7.0 redis常见命令



对redis不熟悉的朋友参考[Redis 教程](#) | [菜鸟教程 \(runoob.com\)](#)

## 7.1 连接池实现代码

见代码：

**mysql\_pool**

redis\_pool

注意测试对应的 参数要和自己实际情况一致，通过宏定义设置

**mysql\_pool**

```
#define DB_HOST_IP      "127.0.0.1"      // 数据库服务器ip
#define DB_HOST_PORT    3306
#define DB_DATABASE_NAME "mysql_pool_test" // 数据库对应的库名字, 这里需要自己提前用命令创建完毕
#define DB_USERNAME     "root"           // 数据库用户名
#define DB_PASSWORD     "123456"         // 数据库密码
#define DB_POOL_NAME     "mysql_pool"     // 连接池的名字, 便于将多个连接池集中管理
#define DB_POOL_MAX_CON  4               // 连接池支持的最大连接数量
```

**redis\_pool**

```
#define DB_HOST_IP      "127.0.0.1"      // 数据库服务器ip
#define DB_HOST_PORT    6379
#define DB_INDEX        0                // redis默认支持16个db
#define DB_PASSWORD     ""               // 数据库密码, 不设置AUTH时该参数为空
#define DB_POOL_NAME     "redis_pool"    // 连接池的名字, 便于将多个连接池集中管理
#define DB_POOL_MAX_CON  4               // 连接池支持的最大连接数量
```

### 7.1.1 msyql\_pool测试

编译：

```
cd mysql_pool
mkdir build
cmake ..
make
```

编译后的执行文件在build目录

**测试数据库的连接**

1. test\_curd.cpp对应源码
2. 修改主机的地址：#define DB\_HOST\_IP "127.0.0.1" // 数据库服务器ip  
1. 以及其他的参数，比如端口、用户名、密码等
3. 执行：./test\_curd

**本地：每个任务都重新创建连接**

1. test\_dbpool.cpp 对应workNoPool函数的执行

2. 修改主机的地址: #define DB\_HOST\_IP "127.0.0.1" // 数据库服务器ip

1. 以及其他的参数, 比如端口、用户名、密码等

3. 执行: ./test\_dbpool 4 4 0

### 本地: 任务从连接池读取连接

1. test\_dbpool.cpp 对应workUsePool函数的执行

2. 修改主机的地址: #define DB\_HOST\_IP "127.0.0.1" // 数据库服务器ip

1. 以及其他的参数, 比如端口、用户名、密码等

3. 执行: ./test\_dbpool 4 4 1

### 远程: 每个任务都重新创建连接

1. test\_dbpool.cpp 对应workNoPool函数的执行

2. 修改主机的地址: #define DB\_HOST\_IP "127.0.0.1" // 数据库服务器ip

1. 以及其他的参数, 比如端口、用户名、密码等

3. 执行: ./test\_dbpool 4 4 0

### 远程: 任务从连接池读取连接

1. test\_dbpool.cpp 对应workNoPool函数的执行

2. 修改主机的地址: #define DB\_HOST\_IP "114.215.169.66" // 数据库服务器ip

1. 以及其他的参数, 比如端口、用户名、密码等

3. 执行: ./test\_dbpool 4 4 1

### 线程数量 连接池 不用连接池

本地虚拟机测试 Ubuntu 16.04 cpu逻辑核数4 插入1000条数据

io密集型 线程池 线程数量  $2 * \text{cpu核数} + (1/2) = 2 * 4 + (1/2) = 9$ , 10个线程

线程数量	连接池1	连接池2	不用连接池1	不用连接池2
1	9366ms 9秒	9199ms	9550ms	9389ms
2	9071ms 9秒	9084ms	8991ms	8994ms
4	4578ms	4611ms	5249ms	5206ms
8	<b>2308ms</b>	2301ms	<b>3182ms</b>	3211ms
16	<b>1316ms</b>	1283ms	<b>2605ms</b>	2945ms
32	<b>910ms</b>	898ms	<b>2580ms</b>	2669ms
64	885ms 小于1秒	912ms	2570ms	2775ms
128	1357ms	1385ms	2684ms	2837ms
256	2355ms	2364ms	3190ms	4311ms

思考：为什么这里在32~64个线程之间效率是最高？

阿里云服务器 5.7.33-0ubuntu0.16.04.1 **cpu逻辑核数1**

线程数量	连接池1	连接池2	不用连接池1	不用连接池2
1	714ms	791ms	14256ms	14577ms
2	589ms	651ms	13368ms	13495ms
4	413ms	432ms	13976ms	12847ms
8	371ms	389ms	13624ms	13505ms
16	453ms	455ms	12874ms	13583ms
32	646ms	765ms	12667ms	13983ms
64	1107ms	1106ms	12643ms	13665ms

## 7.1.2 redis\_pool测试

编译

```
cd mysql_pool
mkdir build
cmake ..
make
```

编译后的执行文件在build目录

**对应test\_CachePool.cpp**

- testCacheUsePool 函数 测试使用同一连接重复set数据
- testCacheOneCmdPerConn函数测试 每次set都重新创建连接

**对应test\_dbpool.cpp**

- ./test\_dbpool 4 4 1 代表使用4线程，4连接，1使用连接池
- ./test\_dbpool 4 4 0 代表使用4线程，4连接，0不用连接池

**单线程** 本地虚拟机测试 Ubuntu 16.04 cpu逻辑核数4

测试次数	同步连接池	不用连接池	同步长连接	异步长连接
1000	77ms	180ms	62ms	2ms

**多线程** 本地虚拟机测试 Ubuntu 16.04 cpu逻辑核数4

线程数量	连接池1	不用连接池1
1	69ms	166ms
2	36ms	82ms
4	22ms	68ms
8	22ms	58ms
16	18ms	48ms
32	21ms	54ms/80ms
64	44ms	53ms/86ms

## 7.2 MySQL和Redis客户端连接编程

具体的mysql、redis客户端编程后续在 mysql专题、redis专题里讲解。

mysql api c客户端: [https://www.yuque.com/linuxer/linux\\_senior/rcz4xl](https://www.yuque.com/linuxer/linux_senior/rcz4xl)

hiredis的使用: [https://www.yuque.com/linuxer/linux\\_senior/ofs3au](https://www.yuque.com/linuxer/linux_senior/ofs3au)

## 7.3 案例

问: 4个连接池对象和4个线程使用4个连接池做同样的事情吗? 还是区分每个线程做不同的事情。

答: 连接池只是提供了连接对象, 提供了一条连接通道, 至于调用者要拿这个连接对象做什么业务是用调用者取决定的。出于业务解耦合的场景, 也可以设置不同的线程池和不同的连接池应对不同的业务, 比如即时通讯写入聊天记录和读取聊天记录采用不同的线程池和对象池。

# 8 连接池连接设置数量

## 8.1 经验公式

连接数 = ((核心数 \* 2) + 有效磁盘数)

按照这个公式, 即是说你的服务器 CPU 是 4核 i7 的, 那连接池连接数大小应该为 ((4\*2)+1)=9

这里只是一个经验公式。还要和线程池数量以及具体业务结合在一起。来自: [About Pool Sizing : brettwooldridge/HikariCP Wiki \(github.com\)](https://brettwooldridge.com/HikariCP-Wiki/)(HikariCP 是一个高性能的 JDBC 连接池组件,号称性能最好的后起之秀)

CPU总核数 = 物理CPU个数 \* 每颗物理CPU的核数

总逻辑CPU数 = 物理CPU个数 \* 每颗物理CPU的核数 \* 超线程数

查看CPU信息（型号）

```
[root@AAA ~]# cat /proc/cpuinfo | grep name | cut -f2 -d: | uniq -c
      4  Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz
```

# 查看物理CPU个数

```
[root@AAA ~]# cat /proc/cpuinfo | grep "physical id" | sort | uniq | wc -l
2
```

# 查看每个物理CPU中core的个数(即核数)

```
[root@AAA ~]# cat /proc/cpuinfo | grep "cpu cores" | uniq
cpu cores      : 2
```

# 查看逻辑CPU的个数

```
[root@AAA ~]# cat /proc/cpuinfo | grep "processor" | wc -l
4
```

## 8.2 IO密集型的任务

如果任务整体上是一个IO密集型的任务。在处理一个请求的过程中（处理一个任务），总共耗时100+5=105ms，而其中只有5ms是用于计算操作的（消耗cpu），另外的100ms等待io响应，CPU利用率为 $5/(100+5)$ 。

使用线程池是为了尽量提高CPU的利用率，减少对CPU资源的浪费，假设以100%的CPU利用率来说，要达到100%的CPU利用率，对于一个CPU就要设置其利用率的倒数个数的线程数，也即 $1/(5/(100+5))=21$ ，4个CPU的话就乘以4。那么算下来的话，就是.....84，这个时候线程池要设置**84个线程数**，然后连接池也是设置为84个连接。

## 9 其他操作参考

### MySQL提示“too many connections”的解决方法

1. 首先登录mysql终端，输入**show variables like max\_connections**，查看最大连接数。

```
mysql> show variables like '%max_connection%';
+-----+-----+
| variable_name | value |
+-----+-----+
| max_connections | 151   |
+-----+-----+
1 row in set (0.19 sec)
```

默认连接数量这里只有151，可以根据自己需要修改。比如可以**临时**设置为1000

```
set GLOBAL max_connections=1000;
```

连接数长期生效参考地址：[mysql数据库Too many connections - 八戒vs - 博客园\(cnblogs.com\)](http://mysql数据库Too many connections - 八戒vs - 博客园(cnblogs.com))

修改长期生效的方式需要 重启mysql

```
sudo service mysql restart
```

## Table 'performance\_schema.session\_variables' doesn't exist

[解决mysql获取连接报Table 'performance\\_schema.session\\_variables' doesn't exist错误! Just code的博客-CSDN博客table'performance\\_schema](#)

## druid连接池

可以研究下druid连接池的设计理念，扩展对连接池的理解。