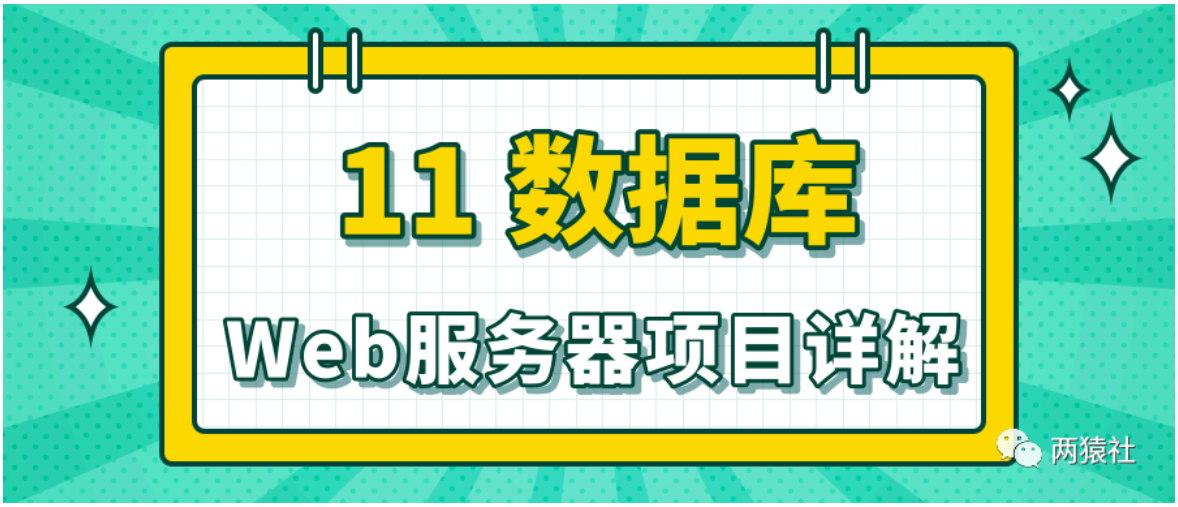
最新版Web服务器项目详解 - 11 数据库连接池

原创 互联网猿 两猿社 2020-05-13 08:28



互联网猿|两猿社

基础知识

什么是数据库连接池？

池是一组资源的集合，这组资源在服务器启动之初就被完全创建好并初始化。通俗来说，池是资源的容器，本质上是对资源的复用。

顾名思义，连接池中的资源为一组数据库连接，由程序动态地对池中的连接进行使用，释放。

当系统开始处理客户请求的时候，如果它需要相关的资源，可以直接从池中获取，无需动态分配；当服务器处理完一个客户连接后,可以把相关的资源放回池中，无需执行系统调用释放资源。

数据库访问的一般流程是什么？

当系统需要访问数据库时，先系统创建数据库连接，完成数据库操作，然后系统断开数据库连接。

为什么要创建连接池？

从一般流程中可以看出，若系统需要频繁访问数据库，则需要频繁创建和断开数据库连接，而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。

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，更加安全可靠。

整体概述

池可以看做资源的容器，所以多种实现方法，比如数组、链表、队列等。这里，使用单例模式和链表创建数据库连接池，实现对数据库连接资源的复用。

项目中的数据库模块分为两部分，其一是数据库连接池的定义，其二是利用连接池完成登录和注册的校验功能。具体的，工作线程从数据库连接池取得一个连接，访问数据库中的数据，访问完毕后将连接交还连接池。

本文内容

本篇将介绍数据库连接池的定义，具体的涉及到单例模式创建、连接池代码实现、RAII机制释放数据库连接。

单例模式创建，结合代码描述连接池的单例实现。

连接池代码实现，结合代码对连接池的外部访问接口进行详解。

RAII机制释放数据库连接，描述连接释放的封装逻辑。

单例模式创建

使用局部静态变量懒汉模式创建连接池。

```
class connection_pool
{
public:
    //局部静态变量单例模式
    static connection_pool *GetInstance();

private:
    connection_pool();
    ~connection_pool();
}

connection_pool *connection_pool::GetInstance()
{
    static connection_pool connPool;
    return &connPool;
}
```

连接池代码实现

连接池的定义中注释比较详细，这里仅对其实现进行解析。

连接池的功能主要有：初始化，获取连接、释放连接，销毁连接池。

初始化

值得注意的是，销毁连接池没有直接被外部调用，而是通过RAII机制来完成自动释放；使用信号量实现多线程争夺连接的同步机制，这里将信号量初始化为数据库的连接总数。

```
connection_pool::connection_pool()
{
    this->CurConn = 0;
    this->FreeConn = 0;
}

//RAII机制销毁连接池
connection_pool::~connection_pool()
{
    DestroyPool();
}

//构造初始化
void connection_pool::init(string url, string User, string PassWord, string DBName
{
    //初始化数据库信息
    this->url = url;
    this->Port = Port;
    this->User = User;
    this->PassWord = PassWord;
    this->DatabaseName = DBName;

    //创建MaxConn条数据库连接
    for (int i = 0; i < MaxConn; i++)
    {
        MYSQL *con = NULL;
        con = mysql_init(con);

        if (con == NULL)
        {
            cout << "Error:" << mysql_error(con);
            exit(1);
        }
        con = mysql_real_connect(con, url.c_str(), User.c_str(), PassWord.c_str(),

        if (con == NULL)
        {
            cout << "Error: " << mysql_error(con);
            exit(1);
        }

        //更新连接池和空闲连接数量
        connList.push_back(con);
        ++FreeConn;
    }

    //将信号量初始化为最大连接次数
    reserve = sem(FreeConn);

    this->MaxConn = FreeConn;
}
```

获取、释放连接

当线程数量大于数据库连接数量时，使用信号量进行同步，每次取出连接，信号量原子减1，释放连接原子加1，若连接池内没有连接了，则阻塞等待。

另外，由于多线程操作连接池，会造成竞争，这里使用互斥锁完成同步，具体的同步机制均使用lock.h中封装好的类。

```
//当有请求时，从数据库连接池中返回一个可用连接，更新使用和空闲连接数
MYSQL *connection_pool::GetConnection()
{
    MYSQL *con = NULL;

    if (0 == connList.size())
        return NULL;

    //取出连接，信号量原子减1，为0则等待
    reserve.wait();

    lock.lock();

    con = connList.front();
    connList.pop_front();

    //这里的两个变量，并没有用到，非常鸡肋...
    --FreeConn;
    ++CurConn;

    lock.unlock();
    return con;
}

//释放当前使用的连接
bool connection_pool::ReleaseConnection(MYSQL *con)
{
    if (NULL == con)
        return false;

    lock.lock();

    connList.push_back(con);
    ++FreeConn;
    --CurConn;

    lock.unlock();

    //释放连接原子加1
    reserve.post();
    return true;
}
```

销毁连接池

通过迭代器遍历连接池链表，关闭对应数据库连接，清空链表并重置空闲连接和现有连接数量。

```
//销毁数据库连接池
void connection_pool::DestroyPool()
{
    lock.lock();
    if (connList.size() > 0)
    {
        //通过迭代器遍历，关闭数据库连接
        list<MYSQL *>::iterator it;
        for (it = connList.begin(); it != connList.end(); ++it)
        {
            MYSQL *con = *it;
            mysql_close(con);
        }
        CurConn = 0;
        FreeConn = 0;

```

```
17          //清空list
18          connList.clear();
19
20          lock.unlock();
21      }
22
23      lock.unlock();
24  }
```

RAII机制释放数据库连接

将数据库连接的获取与释放通过RAII机制封装，避免手动释放。

定义

这里需要注意的是，在获取连接时，通过有参构造对传入的参数进行修改。其中数据库连接本身是指针类型，所以参数需要通过双指针才能对其进行修改。

```
1   class connectionRAII{
2
3   public:
4       //双指针对MYSQL *con修改
5       connectionRAII(MYSQL **con, connection_pool *connPool);
6       ~connectionRAII();
7
8   private:
9       MYSQL *conRAII;
10      connection_pool *poolRAII;
11  };
```

实现

不直接调用获取和释放连接的接口，将其封装起来，通过RAII机制进行获取和释放。

```
1   connectionRAII::connectionRAII(MYSQL **SQL, connection_pool *connPool){
2       *SQL = connPool->GetConnection();
3
4       conRAII = *SQL;
5       poolRAII = connPool;
6   }
7
8   connectionRAII::~connectionRAII(){
9       poolRAII->ReleaseConnection(conRAII);
10  }
```

如果本文对你有帮助， 阅读原文 star一下服务器项目，我们需要你的星星^_^.

完。