**线程池通用库介绍**

# 一、什么是线程池？

诸如web服务器、数据库服务器、文件服务器和邮件服务器等诸多服务器应用都面向处理来自某些远程来源的大量短小的任务。构建服务器应用的一个过于简单的模型是：每当一个请求达到就创建一个新的服务对象，然后在新的服务对象中为请求服务。但当有大量请求并发访问时，服务器不断的创建和销毁对象的开销很大。所以提高服务器效率的一个手段就是尽可能减少创建和销毁对象的次数，特别是一些很耗资源的对象创建和销毁，这样就引入了“池”的概念，“池”的概念使得人们可以定制一定量的资源，然后都这些资源进行复用，而不是频繁的创建和销毁。

线程池是预先创建线程的一种技术，线程池在还没有任务到来之前，创建一定数量的线程，放入空闲队列中。这些线程都是处于睡眠状态，即均为启动，但不销毁CPU，而只是占用较小的内存空间。当请求到来之后，缓冲池给这请求分配一个空闲线程，把请求传入此线程中运行，进行处理。当预先创建的线程都处于运行状态，即预制线程不够，线程池可以自由创建一定数量的新线程，用于处理更多的请求。当系统比较空闲的时候，也可以通过移除一部分一直处于停用状态的线程。

# 二、线程池的注意事项

虽然线程池是构建多线程应用程序的强大机制，但使用它并不是没有风险。在使用线程池时需要注意线程池大小与性能的关系，注意并发风险、死锁、资源不足和线程泄露等问题。

（1）线程池大小：多线程应用并非线程越多越好，需要根据系统运行的软硬件环境以及应用本身的特点决定线程池的大小。一般来说，如果代码结构合理的话，线程数目与CPU数量相适合即可。如果线程运行时可能出现阻塞现象，可相应增加池的大小；如果有必要可采用自适应算法来动态调整线程池的大小，以提高CPU的有效利用率和系统的整体性能。

（2）并发错误：多线程应用需要特别注意并发错误，要从逻辑上保证程序的正确性，注意避免死锁现象的发生。

（3）线程泄露：这是线程池应用中一个严重的问题，当任务执行完毕而线程没能返回池中就发生线程泄露现象。

# 三、简单线程池的设计

一个典型的线程池，应该包括如下几个部分：

1.线程池管理器（ThreadPool），用于启动、停用、管理线程池；

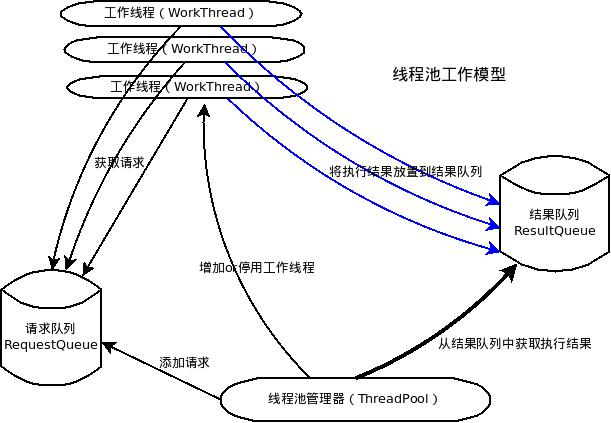
2.工作线程（WorkThread），线程池中的线程；

3.请求接口（WorkRequest），创建请求对象，以供工作线程调度任务的执行；

4.请求队列（RequestQueue），用于存放和提取请求；

5.结果队列（ResultQueue），用于储存请求执行后返回的结果；

线程池管理器，通过添加请求的方法（putRequest）向请求队列（RequestQueue）添加请求，这些请求事先需要实现请求接口，即传递工作函数、参数、结果处理函数、以及异常处理函数。之后初始化一定数量的工作线程，这些线程通过轮询的方式不断查看请求队列（RequestQueue），只要有请求存在，则会提取出请求，进行执行。然后，线程池管理器调用方法（poll）查看结果队列（resultQueue）是否有值，如果有值，则取出，调用结果处理函数执行。通过以上讲述，不难发现，这个系统的核心资源在于请求队列和结果队列，工作线程通过轮询requestQueue获得任务，主线程通过查看结果队列，获得执行结果。因此，对这个队列的设计，要实现线程同步，以及一定阻塞和超时机制的设计，以防止因为不断轮询而导致的过多cpu开销。



# 四、线程池的功能

1.可创建用户指定的固定线程数；

2.可根据业务的运转情况，动态的创建新的线程；

3.可根据业务的运转情况，动态的销毁部分线程；

4.可处理不同优先级的任务请求；

# 五、接口介绍

创建线程池：

int create(int fix\_thread\_num, int dyn\_thread\_num, int max\_queue\_size);

fix\_thread\_num：申请创建固定的工作线程个数；

dyn\_thread\_num：预留的动态工作线程个数；

max\_queue\_size：运转过程中，最大的工作队列数，分为高低优先级，实际是两倍；

销毁线程池：

void destroy(void);

添加工作队列：

int addWork(ThreadPoolPriority priority,void (\*routine)(void \*), void \*arg);

priority：要添加任务的优先级，可取值范围为：

enum ThreadPoolPriority{

LowPriority = 1,

HighPriority = 0,

};

routine：工作任务的执行函数，函数原型必须是void func(void \*arg)类型；

arg：工作任务的数据，也就是传递给routine的参数；

获取当前任务数量：

int getCurQueueSize(void);

获取当前活跃线程数：

int getCurThreadSize(void);