# 背景

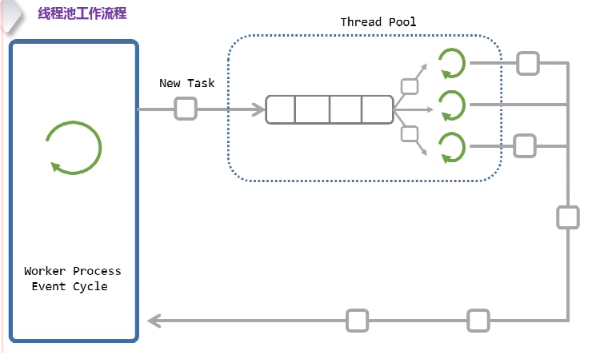
之前所述，select、poll、epoll等IO多路复用都是客户端连接服务器技术，**线程池位于客户端**，即客户端已经完成与服务端的连接，可以使用高效地方法接收数据，接下来就是如何高效地处理数据。

# 原理

提前创建好一堆线程，**客户端连接的时候直接使用，不需要频繁地创建和销毁线程**，提高响应客户端的时间。

客户端发过来的数据对应一个事件，在服务端维护着事件的任务队列（这里使用条件变量判断是否需要启动线程相应事件，即任务队列不为空时线程池调用pthread\_sinal或pthread\_brodcast取任务，否则调用pthread\_cond\_wait阻塞），客户端发过来的事件在服务端会启动相应的线程处理。

线程池主要用于异步解耦，线程池如何做到异步解耦？



注：左侧对应的是工作线程，工作线程将新的任务加入到线程池的任务队列中，右侧线程池包括任务队列和实际的各个线程，各个线程负责任务的执行。所谓的异步解耦，实际上就是任务加入到任务队列和任务的执行是异步解耦的。

## 作用

在开发过程中，合理地利用线程池有以下好处：

1. 降低资源消耗，通过重复利用已创建的线程降低线程创建和销毁造成的消耗；
2. 提高响应速度，当任务到达时，任务可以不需要等到线程创建就能立即执行；
3. 提高线程的可管理性，线程是稀缺资源，如果无限制地创建，不仅会消耗系统资源，还会降低系统的稳定性，使用线程池可以进行统一分配、调优和监控。

## 实现一

线程池如何实现？

1. 任务队列：多个线程共享的临界资源
2. 执行队列：多个执行线程的集合
3. 管理组件：用于管理任务队列和执行队列有秩序的工作

代码：

1. 定义任务队列

// 任务队列

struct NJOB{

void (\*func)(void \*arg); //回调函数

void \*user\_data; //用户信息

struct NJOB \*prev;

struct NJOB \*next;

};

1. 定义执行队列

// 工作队列

struct NWORKER{

pthread\_t id;

struct NWORKER \*prev;

struct NWORKER \*next;

};

1. 定义管理组件

struct NMANAGER{

pthread\_mutex\_t mtx; //每次从任务队列取数据需要加锁

pthread\_cond\_t cond;

//定义线程挂起状态(如果任务队列不存在任务的时候，则线程挂起等待)

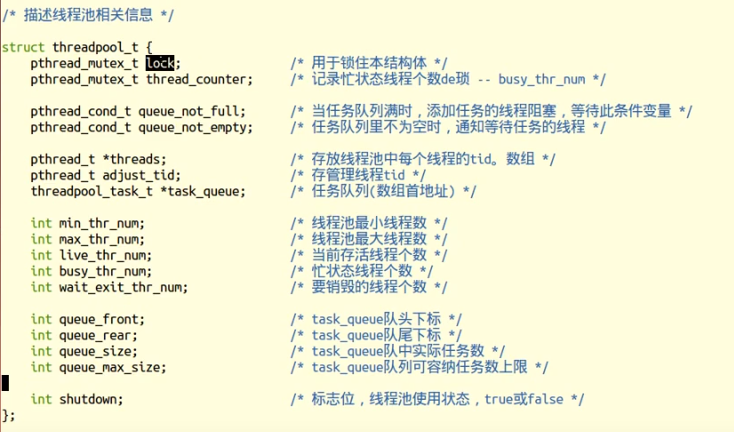
struct NJOB \*jobs;

struct NWORKER \*workers;

};

## 实现二

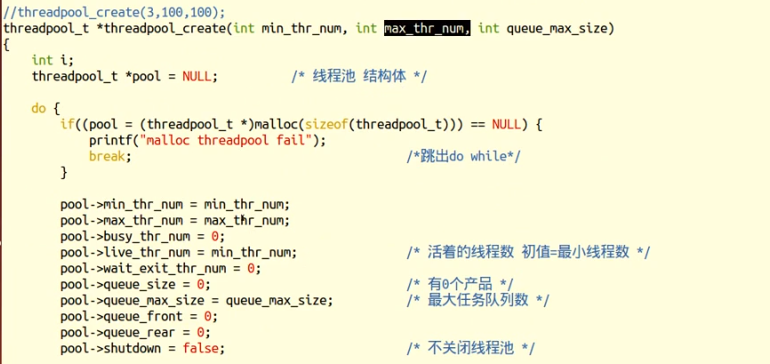
### 线程池结构体



### 创建线程池

基本操作：

1. 创建线程池结构体的指针
2. 初始化线程池结构体（N个成员变量）
3. 创建N个任务线程
4. 创建1个管理者线程
5. 失败时，销毁开辟的所有空间







### 管理线程

### 向线程池中添加任务

# Java线程池

# 应用

Nginx：日志落盘，采用线程池，主要用于异步解耦。