以下是基于 POSIX pthread 的实现，一个可重复使用的线程用于执行带有字符串参数的任务。在 JNI 环境下，这种方式可以用来运行耗时任务，同时参数传递支持字符串。

完整代码示例

JNI 实现代码

```
#include <jni.h>
#include <pthread.h>
#include <string.h>
#include <stdlib.h>
#include <android/log.h>

#define LOG_TAG "JNI_THREAD"
#define LOGI(...)
__android_log_print(ANDROID_LOG_INFO,
LOG_TAG, __VA_ARGS__)

// 全局变量
pthread_mutex_t mutex =
PTHREAD_MUTEX_INITIALIZER; // 互斥锁
pthread_cond_t cond =
PTHREAD_COND_INITIALIZER;    // 条件变量
char* taskArg = NULL;
// 任务参数（字符串）
```

```
int hasTask = 0;
// 是否有任务
int stopThread = 0;
// 停止线程标志

// 工作线程函数
void* workerThread(void* args) {
    while (1) {
        pthread_mutex_lock(&mutex);

        // 等待任务或停止信号
        while (!hasTask && !
stopThread) {
            pthread_cond_wait(&cond,
&mutex);
        }

        // 如果是停止信号，则退出线程
        if (stopThread) {

pthread_mutex_unlock(&mutex);
            break;
        }

        // 获取任务参数
        char* currentArg = taskArg;
        taskArg = NULL;
        hasTask = 0;
```

```
            pthread_mutex_unlock(&mutex);

            // 执行任务逻辑
            if (currentArg) {
                LOGI("Processing task with
argument: %s", currentArg);
                free(currentArg);  // 释放
字符串内存
            }
        }
        return NULL;
}

// 初始化线程
pthread_t initReusableThread() {
    pthread_t thread;
    pthread_create(&thread, NULL,
workerThread, NULL);
    return thread;
}

// 提交任务
void submitTask(const char* arg) {
    pthread_mutex_lock(&mutex);

    // 设置任务参数
    if (taskArg) {
        free(taskArg);  // 清理旧任务
    }
```

```
    taskArg = strdup(arg);  // 复制字符
串
    hasTask = 1;

    // 通知线程
    pthread_cond_signal(&cond);

    pthread_mutex_unlock(&mutex);
}

// 停止线程
void stopReusableThread(pthread_t
thread) {
    pthread_mutex_lock(&mutex);

    // 设置停止标志
    stopThread = 1;

    // 通知线程
    pthread_cond_signal(&cond);

    pthread_mutex_unlock(&mutex);

    // 等待线程退出
    pthread_join(thread, NULL);
}
```

JNI 接口代码

```c
extern "C"
JNIEXPORT jlong JNICALL
Java_com_example_MyNativeLib_startReusableThread(JNIEnv* env, jobject thiz)
{
    pthread_t thread = initReusableThread();
    return (jlong)thread;  // 返回线程句柄
}

extern "C"
JNIEXPORT void JNICALL
Java_com_example_MyNativeLib_submitTask(JNIEnv* env, jobject thiz, jlong threadHandle, jstring taskArg) {
    const char* cTaskArg = env->GetStringUTFChars(taskArg, NULL);
    submitTask(cTaskArg);  // 提交任务
    env->ReleaseStringUTFChars(taskArg, cTaskArg);
}

extern "C"
JNIEXPORT void JNICALL
```

```
Java_com_example_MyNativeLib_stopReusa
bleThread(JNIEnv* env, jobject thiz,
jlong threadHandle) {
    pthread_t thread =
(pthread_t)threadHandle;
    stopReusableThread(thread);
}
```

Java 层调用代码

```
public class MyNativeLib {
    static {

System.loadLibrary("my_native_lib");
    }

    public native long
startReusableThread();
    public native void submitTask(long
threadHandle, String taskArg);
    public native void
stopReusableThread(long threadHandle);

    public static void main(String[]
args) {
        MyNativeLib lib = new
MyNativeLib();
        long threadHandle =
lib.startReusableThread();
```

```
        lib.submitTask(threadHandle,
"Task 1: Hello JNI!");
        try { Thread.sleep(2000); }
catch (InterruptedException ignored)
{}

        lib.submitTask(threadHandle,
"Task 2: Another string!");
        try { Thread.sleep(2000); }
catch (InterruptedException ignored)
{}


lib.stopReusableThread(threadHandle);
        System.out.println("Reusable
thread stopped.");
    }
}
```

运行结果

假设在 Android 的 logcat 中查看日志输出：

```
I/JNI_THREAD: Processing task with
argument: Task 1: Hello JNI!
I/JNI_THREAD: Processing task with
argument: Task 2: Another string!
```

注意事项

1. **线程安全**：通过 pthread_mutex 和 pthread_cond 确保任务提交和执行的线程安全。
2. **任务参数管理**：字符串参数使用 strdup 复制，避免内存泄漏或非法访问。
3. **线程停止**：使用 stopThread 标志确保线程能安全退出。
4. **JNI 字符串处理**：GetStringUTFChars 和 ReleaseStringUTFChars 用于管理 Java 字符串与 C 字符串的转换。

这是一种简单高效的方式，可以在 JNI 环境下实现一个可重复使用的线程，并传递字符串参数进行任务处理。