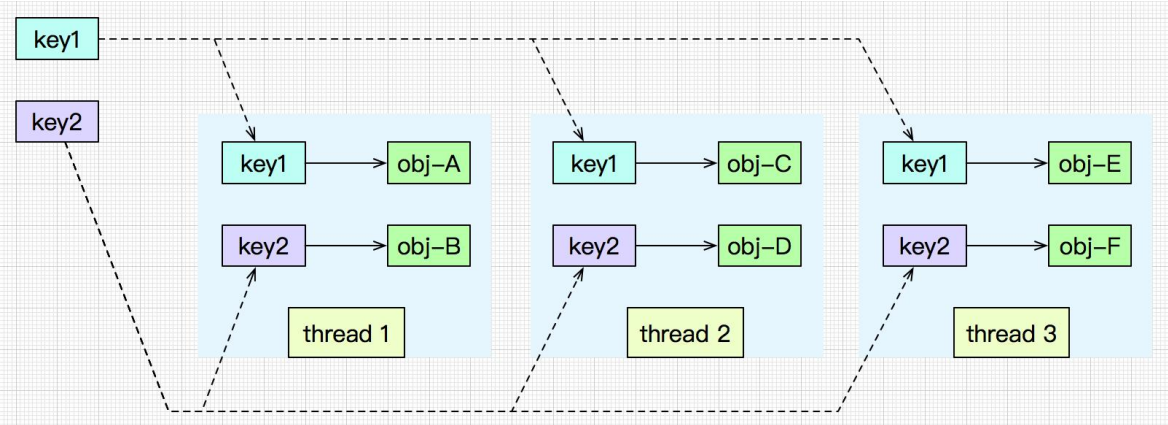
作者：代兴　邮箱：503268771@qq.com　Vx 公众号：东方架构师
https://github.com/drink-cat/Book_Program-Principles

threadlocal 的含义和示意图

threadlocal 表示每个线程有独立的变量空间，实现线程隔离。
threadlocal 是提高并发、优化性能的重要方式。每个线程操作自己的 threadlocal 变量，不需要互斥锁。

threadlocal 的示意图：



threadlocal 的实现过程：每个线程有自己的 threadlocal 空间，从 threadlocal 空间查询 threadlocal 变量；使用 key 表示一个 threadlocal 变量，线程用 key 查询自己的 threadlocal 变量；threadlocal 变量在多个线程中创建多个变量副本，支持懒加载，使用时才新建；操作 threadlocal 变量在单个线程内部，因此不需要加锁。

Java 语言的 threadlocal

Thread 类，包含属性 `ThreadLocal.ThreadLocalMap threadLocals` ，存储 threadlocal 变量。
ThreadLocal 类，包含方法 `public T get()` ，先找到当前的线程，然后找到线程的 threadlocal 变量集合，以 ThreadLocal 自身对象作为 key，在集合中找到对应的 threadlocal 变量。

Thread 线程类：

```
public
class Thread implements Runnable {
    /* ThreadLocal values pertaining to this thread. This map is maintained
     * by the ThreadLocal class. */
    ThreadLocal.ThreadLocalMap threadLocals = null;
```

ThreadLocal 类：

```
public class ThreadLocal<T> {
    /**
     * Returns the value in the current thread's copy of this
     * thread-local variable.  If the variable has no value for the
     * current thread, it is first initialized to the value returned
```

```
     * by an invocation of the {@link #initialValue} method.
     *
     * @return the current thread's value of this thread-local
     */
    public T get() {
        Thread t = Thread.currentThread();
        ThreadLocalMap map = getMap(t);
        if (map != null) {
            ThreadLocalMap.Entry e = map.getEntry(this);
            if (e != null) {
                @SuppressWarnings("unchecked")
                T result = (T)e.value;
                return result;
            }
        }
        return setInitialValue();
    }

    /**
     * Get the map associated with a ThreadLocal. Overridden in
     * InheritableThreadLocal.
     *
     * @param  t the current thread
     * @return the map
     */
    ThreadLocalMap getMap(Thread t) {
        return t.threadLocals;
    }
```

使用 C 实现自定义的 threadlocal

编写代码： threadlocal.c

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <stdint.h>
#include <syscall.h>

// 一个 threadlocal 变量的标识
pthread_key_t key_count;

// 销毁 threadlocal 变量的回调
void key_destroy_callback(void *arg)
{
    int tid = syscall(SYS_gettid);
```

```
    printf(" key_destroy  thread = %d  arg_addr = %p \n", tid, arg);
    free(arg);
}

// 获得 threadlocal 变量。没有就新建。
uint64_t *get_threadlocal_count()
{
    // 获得 threadlocal 变量
    uint64_t *ptr = pthread_getspecific(key_count);
    if (NULL == ptr)
    {
        // 每个变量申请新的内存
        ptr = malloc(sizeof(uint64_t));
        *ptr = 0;
        // 设置 threadlocal 变量
        pthread_setspecific(key_count, ptr);
    }
    return ptr;
}

void *thread_func(void *param)
{
    // 循环操作多次
    for (int i = 0; i < 90000000; ++i)
    {
        // 操作 threadlocal 变量
        uint64_t *ptr = get_threadlocal_count();
        ++(*ptr);
    }

    // 打印结果
    int tid = syscall(SYS_gettid);
    uint64_t *ptr = get_threadlocal_count();
    printf(" thread = %d  count_addr = %p  count_value = %llu \n",
           tid, ptr, *ptr);
    sleep(1);
    return NULL;
}

int main()
{
    // 创建 key
    pthread_key_create(&key_count, key_destroy_callback);
    printf(" key_count = %u \n", key_count);

    // 创建多个线程
    pthread_t t1;
    pthread_create(&t1, NULL, thread_func, NULL);
    pthread_t t2;
```

```
    pthread_create(&t2, NULL, thread_func, NULL);
    pthread_t t3;
    pthread_create(&t3, NULL, thread_func, NULL);

    pthread_join(t1, NULL);
    pthread_join(t2, NULL);
    pthread_join(t3, NULL);
    pthread_key_delete(key_count);
    sleep(1);
    return 0;
}
```

编译代码：

```
gcc threadlocal.c -lpthread -std=gnu99 -o threadlocal
```

运行代码：

```
[root@local threadlocal]# ./threadlocal
 key_count = 0
 thread = 55269  count_addr = 0x7f90340008c0  count_value = 90000000
 thread = 55270  count_addr = 0x7f902c0008c0  count_value = 90000000
 thread = 55271  count_addr = 0x7f90300008c0  count_value = 90000000
 key_destroy  thread = 55270  arg_addr = 0x7f902c0008c0
 key_destroy  thread = 55269  arg_addr = 0x7f90340008c0
 key_destroy  thread = 55271  arg_addr = 0x7f90300008c0
```

分析结果：

3 个线程，分别更新自己的 threadlocal 变量，变量的结果都为 90000000。

定义一个 threadlocal 变量，key 为 `pthread_key_t key_count` ，value 为 `uint64_t *ptr` 。

使用 key 表示 threadlocal 变量，每个线程都用同一个 key 获得自己的 threadlocal 变量，相互隔离。

使用函数 `pthread_getspecific(key_count)` 获得 threadlocal 变量。

使用函数 `pthread_setspecific(key_count, ptr)` 设置 threadlocal 变量。

创建 3 个线程，每个线程使用 key_count 获得自己的 threadlocal 变量，然后更新变量的值。

线程读写 threadlocal 变量，没有加锁，不需要互斥锁。