如何判断线程是否存活

典型回答

判断线程是否存活可以通过Thread下的isAvalive()方法

```
public class Test{

    public static void main(String[] args) throws InterruptedException {
        Thread t1 = new Thread(() -> {
            System.out.println("t1 begin");
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
            }
            System.out.println("t1 end");
        });
        t1.start();
        System.out.println("t1.isAlive()="+t1.isAlive());
        t1.join();
        System.out.println("t1.isAlive()="+t1.isAlive());
    }
}
```

运行结果：

```
t1 begin
t1.isAlive()=true
t1 end
t1.isAlive()=false
```

扩展知识

以下代码执行后t1的isAlive()方法返回是否为true?

```
public class Test{
    public static void main(String[] args) {
        Thread t1 = new Thread(() -> {
```

```
            System.out.println("t1 begin");
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
            }
            System.out.println("t1 end");
        });
        Thread t2 = new Thread(() -> {
            synchronized (t1) {
                System.out.println("t2 begin");
                try {
                    Thread.sleep(2000);
                } catch (InterruptedException e) {
                }
                System.out.println("t2 end");
                System.out.println("t1 isAlive:" + t1.isAlive());
            }

        });
        t1.start();
        t2.start();
    }
}
```

运行结果：

```
t2 begin
t1 begin
t1 end
t2 end
t1 isAlive:true
```

可以看到t1已经结束了，但t1的isAlive()方法返回的是true。
产生这个现象的原因是isAlive()需要拿到当前对象的锁，注意上面代码中t2里对t1对象进行了synchronized，即t1线程在结束时需要修改自己的状态，而t1的被t2锁住，所以无法修改状态，导致isAlive()返回true。
上述具体原因，我们可以看java.lang.Thread.isAlive()方法的实现。

```
  public final native boolean isAlive();
```

可以看到这是个本地方法，对应到jdk源码中`java_lang_Thread::is_alive`方法调用。其底层实现是取当前线程对象中_eetop_offset的值。不为空则返回true。

```
bool java_lang_Thread::is_alive(oop java_thread) {
  JavaThread* thr = java_lang_Thread::thread(java_thread);
  return (thr != NULL);
}
JavaThread* java_lang_Thread::thread(oop java_thread) {
  return (JavaThread*)java_thread->address_field(_eetop_offset);
}
```

_eetop_offset会在调用java.lang.Thread.start()方法时，在jdk源码中，通过`native_thread->prepare(jthread)`的prepare方法设置为当前线程对象

```
void JavaThread::prepare(jobject jni_thread, ThreadPriority prio) {
  ... //此处缩略其他无关代码
  java_lang_Thread::set_thread(thread_oop(), this);//thread_oop()返回了jdk包装好的当前线程oop对象，this传入当前线程对象的地址
 ... //此处缩略其他无关代码
}
void java_lang_Thread::set_thread(oop java_thread, JavaThread* thread) {
  java_thread->address_field_put(_eetop_offset, (address)thread); //设置_eetop_offset值为当前线程对象的地址
}
```

java线程结束时，jvm会调用`JavaThread::exit`方法

```
void JavaThread::exit(bool destroy_vm, ExitType exit_type) {
  ... //此处缩略其他无关代码
  ensure_join(this); //这里开始设置线程状态，包括设置_eetop_offset值为空
  ... //此处缩略其他无关代码
}

static void ensure_join(JavaThread* thread) {
  Handle threadObj(thread, thread->threadObj());
  assert(threadObj.not_null(), "java thread object must exist");
  ObjectLocker lock(threadObj, thread); //重点，拿到当前线程对象的锁
  thread->clear_pending_exception();
  java_lang_Thread::set_thread_status(threadObj(),
java_lang_Thread::TERMINATED); //标记当前线程状态为终止
```

```
  java_lang_Thread::set_thread(threadObj(), NULL); //重点，_eetop_offset值为空
  lock.notify_all(thread);
  thread->clear_pending_exception();
}
```

`ObjectLocker`就是传说中的synchronized的实现，其构造函数如下

```
ObjectLocker::ObjectLocker(Handle obj, Thread* thread, bool doLock) {
  _dolock = doLock;
  _thread = thread;
  debug_only(if (StrictSafepointChecks) _thread-
>check_for_valid_safepoint_state(false);)
  _obj = obj; //拿到当前对象，在上面例子里，obj为t1

  if (_dolock) {
    TEVENT (ObjectLocker) ;
    ObjectSynchronizer::fast_enter(_obj, &_lock, false, _thread); //操作当前对象的对象头，执行synchronized的流程。由于t1的锁被t2持有，所以t1无法设置_eetop_offset值为空。自然isAlive()方法返回的是true，当然线程状态也无法修改为TERMINATED终止。
  }
}
```

总结

以上就是java如何判断线程存活的整个流程：

1. Thread.start()启动线程时会设置_eetop_offset
2. Thread.isAlive()会读取该字段
3. 在线程结束时，jvm会设置该字段为空

ps:线上代码可不要用Thread对象当做对象锁。