栈轨迹通常是通过从异常对象创建点展开栈获得的。有时候需要得到异常抛出点的栈轨迹。异常对象创建点和异常抛出点的位置可能是不同的。可以创建一个异常对象并把它传递给其他线程抛出。（甚至也没有什么能阻止你把异常传递到其他线程，尽管这通常是一个坏实践。）这两种情况意味着不同的问题。如果允许在异常抛出时获得栈轨迹，那么就有机会惰性创建异常对象。在 JVM 中，栈轨迹是强制性保存在异常对象中的，在异常对象构造器中生成。

很多情况下，异常处理器并不真正需要异常对象本身，而是利用异常抛出机制来实现控制流操纵或捕获例外的运行条件。在这些情况中，异常对象只用于帮助找到匹配的异常处理器。一旦找到异常处理器，异常对象实际上就死掉了。为了匹配异常处理器，VM 需要的实际上是异常对象的类，而不是这个对象本身。基于这个观察结果，某些情况下有可能省略异常对象创建，因此也就没有关联的栈轨迹创建。这大大节省了运行时操作，更不要提这些对象创建可能引发的垃圾回收了。

一个解决方案是惰性创建异常对象。也就是说，默认情况下，VM 只在下列情况之一发生的前提下才创建异常对象。

- 情况 1：异常对象构造器的执行会可能会有副作用，比如写入异常对象以外的其他对象，抛出异常，或者进入 monitor。
- 情况 2：目标异常处理器会访问异常对象。
- 情况 3：栈展开过程在到达匹配的异常处理器之前会遇到本地方法帧。

在上面的情况 1 中，必须像平常那样创建异常对象。也就是说，在异常抛出时就要及早创建。在另外两种情况下，VM 可以及早或惰性创建异常对象。惰性创建意味着 VM 可以把创建推迟，直到它找到匹配的异常处理器或者栈展开遇到本地帧时。否则，可以忽略异常对象。需要考虑情况 3 是因为，VM 不知道本地方法会如何处置异常对象。

为了生成栈轨迹，从保存在异常对象中的执行上下文开始执行栈展开过程。它可以从运行时辅助建立的本地帧开始，也可以从引发硬件异常的 Java 帧开始。可以通过栈基指针（对于 Java 帧）或者 Java 簇指针（对于本地帧）识别一个帧。我们使用指令指针作为一个标志来指示抛出上下文的帧类型。

```
Frame_context* start_frame(VM_Thread* thread)
{
    Registers* regs = thread->context_regs;
    Frame_context* frame = vm_alloc(sizeof(Frame_context));

    frame->jcp = thread->jcp;
    frame->eip = regs->eip;

    // 这里 eip 值复用作为一个标志
    if( regs->eip != 0xFFFFFF )
        frame->ebp = regs->ebp; // Java 代码中的硬件异常

    return frame;
}
```