

xiaolincoding.com

4.5 如何避免预读失效和缓存污染的问题？

小林coding

15-18 minutes

5.7 线程崩溃了，进程也会崩溃吗？

来源：公众号@码海

原文地址：[美团一面：线程崩溃了，进程也会崩溃吗？](#)

大家好，我是小林。

之前分享这篇文章的时候：[进程和线程基础知识全家桶，30 张图一套带走](#)，提到说线程的一个缺点：

线程的缺点：

- 当进程中的一个线程崩溃时，会导致其所属进程的所有线程崩溃（这里是针对 C/C++ 语言，Java 语言中的线程奔溃不会造成进程崩溃）。

举个例子，对于游戏的用户设计，则不应该使用多线程的方式，否则一个用户挂了，会影响其他同个进程的线程。

很多同学就好奇，为什么 C/C++ 语言里，线程崩溃后，进程也会崩溃，而 Java 语言里却不会呢？

刚好看到朋友（[公众号：码海](#)）写了一篇：「美团面试

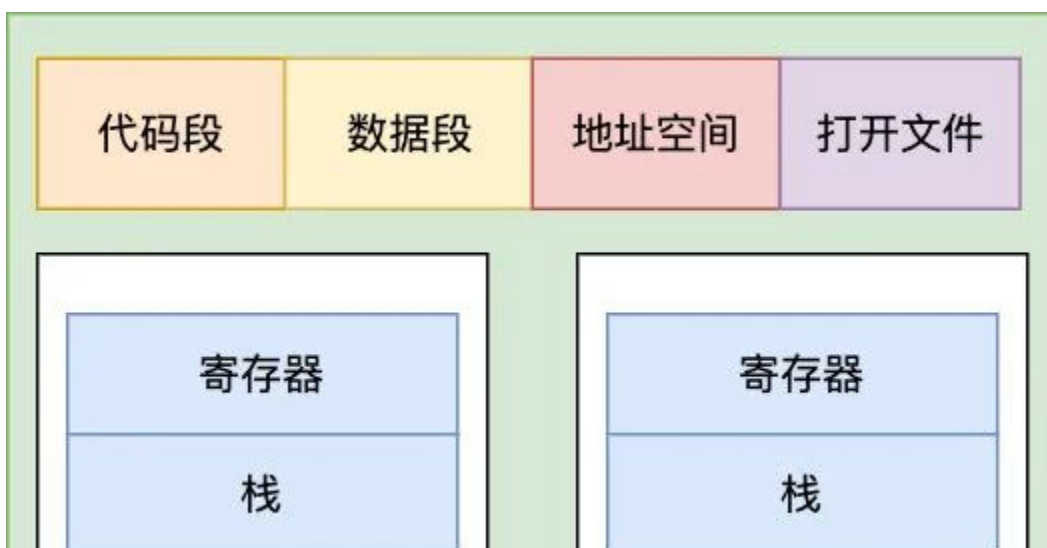
题：为什么线程崩溃不会导致 JVM 崩溃？」

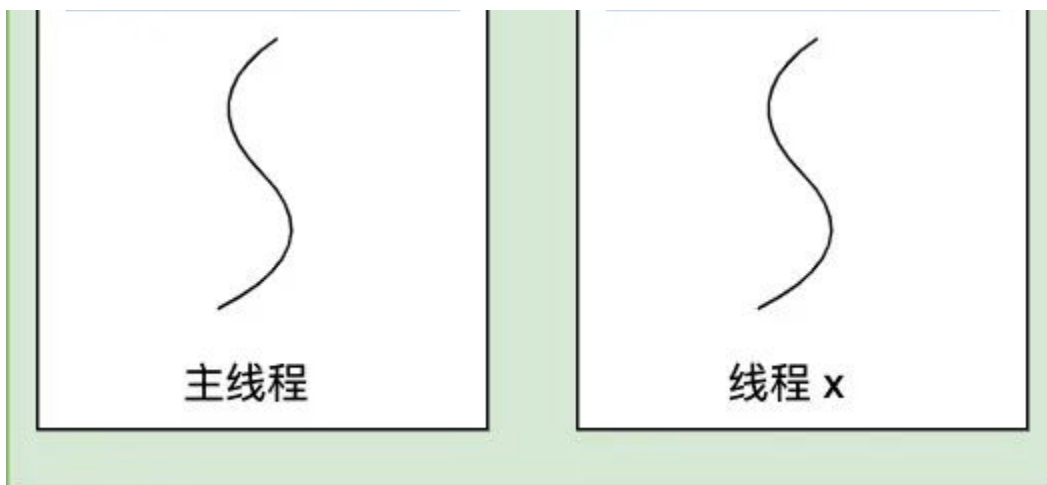
我觉得写的很好，所以分享给大家一起拜读拜读，本文分以下几节来探讨：

1. 线程崩溃，进程一定会崩溃吗
2. 进程是如何崩溃的-信号机制简介
3. 为什么在 JVM 中线程崩溃不会导致 JVM 进程崩溃
4. openJDK 源码解析

线程崩溃，进程一定会崩溃吗

一般来说如果线程是因为非法访问内存引起的崩溃，那么进程肯定会崩溃，为什么系统要让进程崩溃呢，这主要是因为是在进程中，**各个线程的地址空间是共享的**，既然是共享，那么某个线程对地址的非法访问就会导致内存的不确定性，进而可能会影响到其他线程，这种操作是危险的，操作系统会认为这很可能导致一系列严重的后果，于是干脆让整个进程崩溃





线程共享代码段，数据段，地址空间，文件非法访问内存有以下几种情况，我们以 C 语言举例来看看。

- 1、针对只读内存写入数据
- 2、访问了进程没有权限访问的地址空间（比如内核空间）

在 32 位虚拟地址空间中，p 指向的是内核空间，显然不具有写入权限，所以上述赋值操作会导致崩溃

- 3、访问了不存在的内存，比如：

以上错误都是访问内存时的错误，所以统一会报 Segment Fault 错误（即段错误），这些都会导致进程崩溃

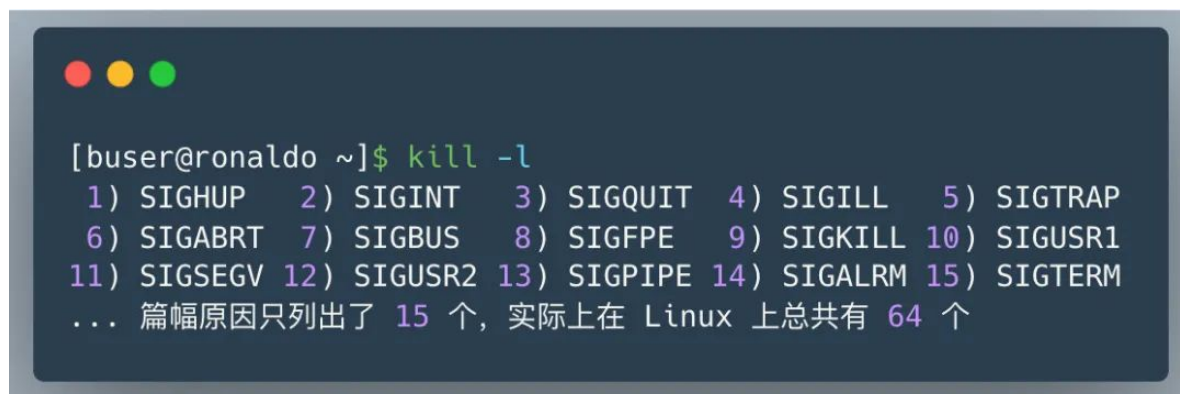
进程是如何崩溃的-信号机制简介

那么线程崩溃后，进程是如何崩溃的呢，这背后的机制到底是怎样的，答案是**信号**。

大家想想要干掉一个正在运行的进程是不是经常用 kill -9

pid 这样的命令，这里的 kill 其实就是给指定 pid 发送终止信号的意思，其中的 9 就是信号。

其实信号有很多类型的，在 Linux 中可以通过 `kill -l` 查看所有可用的信号：



```
[buser@ronaldo ~]$ kill -l
1) SIGHUP    2) SIGINT    3) SIGQUIT   4) SIGILL    5) SIGTRAP
6) SIGABRT   7) SIGBUS    8) SIGFPE    9) SIGKILL   10) SIGUSR1
11) SIGSEGV  12) SIGUSR2  13) SIGPIPE  14) SIGALRM  15) SIGTERM
... 篇幅原因只列出了 15 个，实际上在 Linux 上总共有 64 个
```

当然了发 kill 信号必须具有一定的权限，否则任意进程都可以通过发信号来终止其他进程，那显然是不合理的，实际上 kill 执行的是系统调用，将控制权转移给了内核（操作系统），由内核来给指定的进程发送信号。那么发个信号进程怎么就崩溃了呢，这背后的原理到底是怎样的？

其背后的机制如下

1. CPU 执行正常的进程指令
2. 调用 kill 系统调用向进程发送信号
3. 进程收到操作系统发的信号，CPU 暂停当前程序运行，并将控制权转交给操作系统
4. 调用 kill 系统调用向进程发送信号（假设为 11，即 SIGSEGV，一般非法访问内存报的都是这个错误）

5. 操作系统根据情况执行相应的信号处理程序（函数），一般执行完信号处理程序逻辑后会让进程退出

注意上面的第五步，如果进程没有注册自己的信号处理函数，那么操作系统会执行默认的信号处理程序（一般最后会让进程退出），但如果注册了，则会执行自己的信号处理函数，这样的话就给了进程一个垂死挣扎的机会，它收到 kill 信号后，可以调用 `exit()` 来退出，**也可以使用 `sigsetjmp`，`siglongjmp` 这两个函数来恢复进程的**执行

如代码所示：注册信号处理函数后，当收到 `SIGSEGV` 信号后，先执行相关的逻辑再退出

另外当进程接收信号之后也可以不定义自己的信号处理函数，而是选择忽略信号，如下

也就是说虽然给进程发送了 kill 信号，但如果进程自己定义了信号处理函数或者无视信号就有机会逃出生天，当然了 `kill -9` 命令例外，不管进程是否定义了信号处理函数，都会马上被干掉。

说到这大家是否想起了一道经典面试题：**如何让正在运行的 Java 工程的优雅停机？**

通过上面的介绍大家不难发现，其实是 JVM 自己定义了信号处理函数，这样当发送 `kill pid` 命令（默认会传 15 也就是 `SIGTERM`）后，JVM 就可以在信号处理函数中执行一些资源清理之后再调用 `exit` 退出。

这种场景显然不能用 kill -9，不然一下把进程干掉了资源就来不及清除了。

为什么线程崩溃不会导致 JVM 进程崩溃

现在我们再来看看开头这个问题，相信你多少会心中有数，想想看在 Java 中有哪些是常见的由于非法访问内存而产生的 Exception 或 error 呢，常见的是大家熟悉的 StackoverflowError 或者

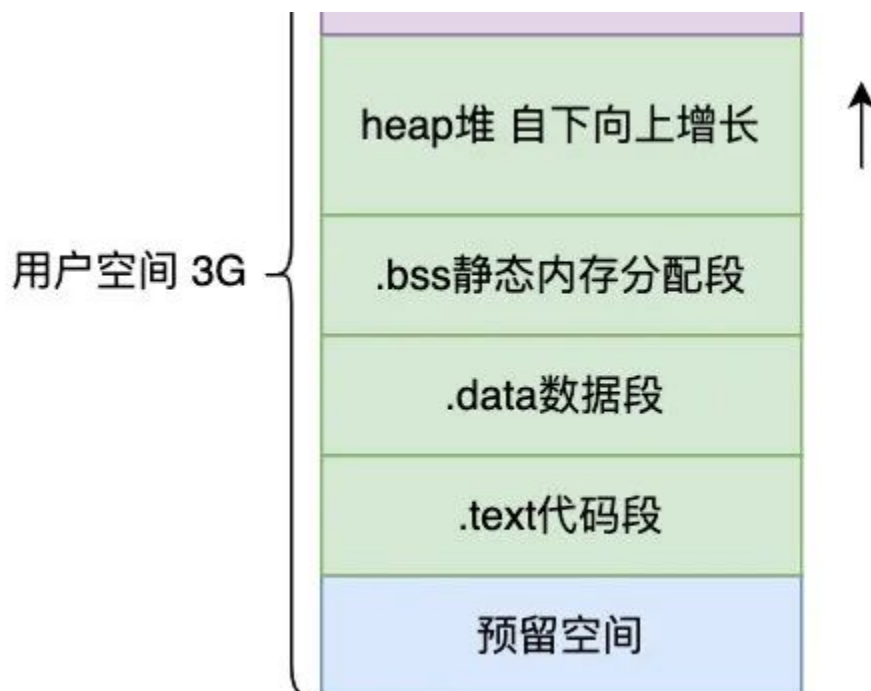
NPE (NullPointerException), NPE 我们都了解，属于是访问了不存在的内存。

但为什么栈溢出 (Stackoverflow) 也属于非法访问内存呢，这得简单聊一下进程的虚拟空间，也就是前面提到的共享地址空间。

现代操作系统为了保护进程之间不受影响，所以使用了虚拟地址空间来隔离进程，进程的寻址都是针对虚拟地址，每个进程的虚拟空间都是一样的，而线程会共用进程的地址空间。

以 32 位虚拟空间，进程的虚拟空间分布如下：

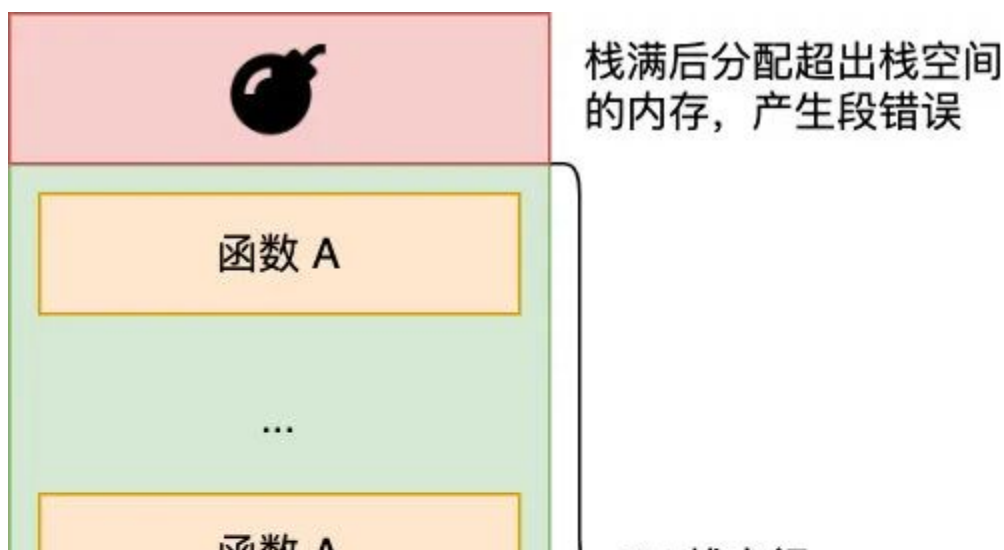


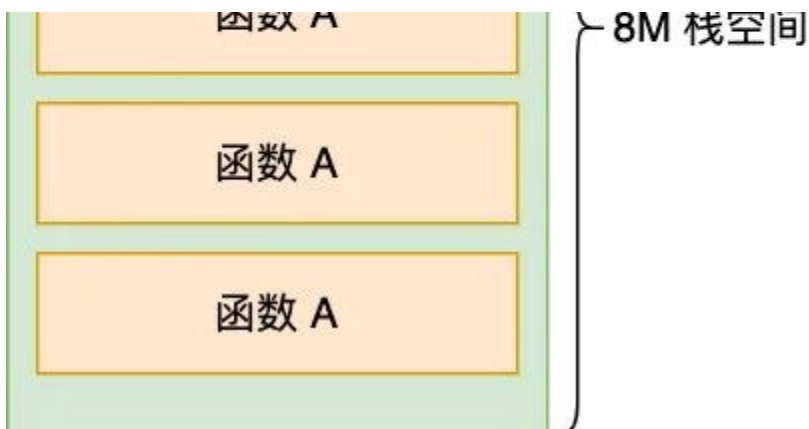


那么 stackoverflow 是怎么发生的呢？

进程每调用一个函数，都会分配一个栈帧，然后在栈帧里会分配函数里定义的各种局部变量。

假设现在调用了一个无限递归的函数，那就会持续分配栈帧，但 stack 的大小是有限的（Linux 中默认为 8 M，可以通过 `ulimit -a` 查看），如果无限递归很快栈就会分配完了，此时再调用函数试图分配超出栈的大小内存，就会发生段错误，也就是 `stackoverflowError`。





好了，现在我们知道了 `StackoverflowError` 怎么产生的。

那问题来了，既然 `StackoverflowError` 或者 `NPE` 都属于非法访问内存，JVM 为什么不会崩溃呢？

有了上一节的铺垫，相信你不难回答，其实就是因为 **JVM 自定义了自己的信号处理函数，拦截了 `SIGSEGV` 信号，针对这两者不让它们崩溃。**

怎么证明这个推测呢，我们来看下 JVM 的源码来一探究竟

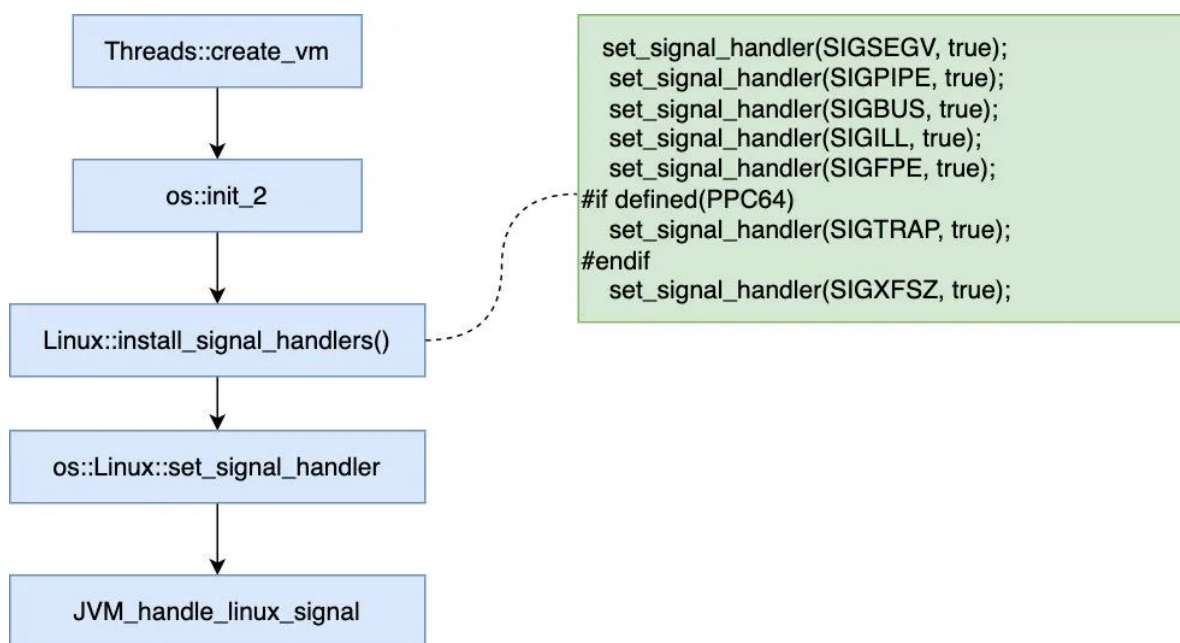
openJDK 源码解析

HotSpot 虚拟机目前使用范围最广的 Java 虚拟机，据 R 大所述，Oracle JDK 与 OpenJDK 里的 JVM 都是 HotSpot VM，从源码层面说，两者基本上是同一个东西。

OpenJDK 是开源的，所以我们主要研究下 Java 8 的 OpenJDK 即可，地址如下：<https://github.com>

</AdoptOpenJDK/openjdk-jdk8u>，有兴趣的可以下载来看看。

我们只要研究 Linux 下的 JVM，为了便于说明，也方便大家查阅，我把其中关于信号处理的关键流程整理了下（忽略其中的次要代码）。



可以看到，在启动 JVM 的时候，也设置了信号处理函数，收到 `SIGSEGV`，`SIGPIPE` 等信号后最终会调用 `JVM_handle_linux_signal` 这个自定义信号处理函数，再来看一下这个函数的主要逻辑。

从以上代码我们可以知道以下信息：

1. 发生 `stackoverflow` 还有空指针错误，确实都发送了 `SIGSEGV`，只是虚拟机不选择退出，而是自己内部作了额外的处理，其实是恢复了线程的执行，并抛出 `StackoverflowError` 和 `NPE`，这就是为什么 JVM 不会崩溃且我们能捕获这两个错误/异常的原因

2. 如果针对 SIGSEGV 等信号，在以上的函数中 JVM 没有做额外的处理，那么最终会走到 `report_and_die` 这个方法，这个方法主要做的事情是生成 `hs_err_pid_xxx.log` crash 文件（记录了一些堆栈信息或错误），然后退出至此我相信大家明白了为什么发生了 `StackoverflowError` 和 `NPE` 这两个非法访问内存的错误，JVM 却没有崩溃。

原因其实就是虚拟机内部定义了信号处理函数，而在信号处理函数中对这两者做了额外的处理以让 JVM 不崩溃，另一方面也可以看出如果 JVM 不对信号做额外的处理，最后会自己退出并产生 crash 文件

`hs_err_pid_xxx.log`（可以通过 `-XX:ErrorFile=/var/log/hs_err.log` 这样的方式指定），这个文件记录了虚拟机崩溃的重要原因。

所以也可以说，虚拟机是否崩溃只要看它是否会产生此崩溃日志文件

总结

正常情况下，操作系统为了保证系统安全，所以针对非法内存访问会发送一个 SIGSEGV 信号，而操作系统一般会调用默认的信号处理函数（一般会让相关的进程崩溃）。

但如果进程觉得“罪不致死”，那么它也可以选择自定义

一个信号处理函数，这样的话它就可以做一些自定义的逻辑，比如记录 crash 信息等有意义的事。

回过头来看为什么虚拟机会针对 `StackoverflowError` 和 `NullPointerException` 做额外处理让线程恢复呢，针对 `stackoverflow` 其实它采用了一种栈回溯的方法保证线程可以一直执行下去，而捕获空指针错误主要是这个错误实在太普遍了。

为了这一个很常见的错误而让 JVM 崩溃那线上的 JVM 要宕机多少次，所以出于工程健壮性的考虑，与其直接让 JVM 崩溃倒不如让线程起死回生，并且将这两个错误/异常抛给用户来处理。

哈喽，我是小林，就爱图解计算机基础，如果觉得文章对你有帮助，欢迎微信搜索「小林coding」，关注后，回复「网络」再送你图解网络 PDF



扫一扫，关注「小林coding」公众号

图解计算机基础
认准**小林coding**

每一张图都包含小林的认真
只为帮助大家能更好的理解

- ① 关注公众号回复「**图解**」
获取图解系列 PDF
- ② 关注公众号回复「**加群**」
拉你进百人技术交流群

上次更新: 6/16/2022, 2:33:07 PM

