空间中的单词...

一个关于生命、宇宙和其他一切的博客

Debugging the JDK with GDB
使用 GDB 调试 JDK

Posted by neugens on February 26, 2015

发布者 neugens 上月26,2015

At Fosdem Volker presented a very great session on how to debug **OpenJDK** (and Hotspot) with **gdb**. Roman and Andrew (Dinn) did something similar while speaking about Shenandoah. In the next few days I'll try to upload their slides on the FOSDEM website so that anyone can access that (and hopefully we will have the recordings this time as well).

在 Fosdem 大会上，Volker 展示了一个非常棒的会议，介绍了如何使用 **gdb** 调试 **OpenJDK**（和 Hotspot）。罗曼和安德鲁 （Dinn） 在谈到 Shenandoah 时也做了类似的事情。在接下来的几天里，我将尝试将他们的幻灯片上传到 FOSDEM 网站上，以便任何人都可以访问它（希望我们这次也能获得录音）。

There are a few things though that I keep forgetting myself and so I thought it would be useful to sum up in a blog post, hopefully general enough for most people as well as future reference for myself!

不过，有几件事我总是忘记自己，所以我认为在一篇博文中总结会很有用，希望对大多数人来说足够通用，以及我自己的未来参考！

Suppose you are trying to detect some funny behaviour in your code, and that the crash is in a native library (or perhaps in some native OpenJDK code which is not hotspot).

假设您正在尝试检测代码中的一些奇怪行为，并且崩溃发生在本机库中（或者可能在一些不是热点的本机 OpenJDK 代码中）。

What you would usually do with Java code is to start your debugger in Eclipse or IntelliJ or whatever, and go step by step until you figure out what's wrong.

通常，使用 Java 代码时，会在 Eclipse 或 IntelliJ 或其他任何语言中启动调试器，然后逐步进行，直到找出问题所在。

But when dealing with native code the thing gets complex, Eclipse and NetBeans can't follow by default the native code, IntelliJ doesn't even support native code at all (at least on Linux). There is an option though, first, you can still use those tools in process attach mode, they have very good debugging interfaces that make it easier to analyse quickly anything, but you can also use gdb directly, likewise in process attach mode.

但是在处理本机代码时，事情会变得复杂，Eclipse 和 NetBeans 默认不能遵循本机代码，IntelliJ 甚至根本不支持本机代码（至少在 Linux 上是这样）。不过有一个选项，首先，你仍然可以在进程附加模式下使用这些工具，它们有非常好的调试界面，可以更轻松地快速分析任何内容，但你也可以直接使用 gdb，同样在进程附加模式下。

Let's see a couple of common cases here:

让我们看看这里的几个常见情况：

1. The application crashes, you want gdb launched automagically:

1. 应用程序崩溃，您希望自动启动 gdb：

```
$ java -XX:OnError="gdb - %p" MyApplication
```

Roman (https://rkennke.wordpress.com/2008/02/29/debugging-native-code-with-hotspot-and-gdb/) (thanks!) show me this trick back in 2008! Honestly, I didn't test that recently, but I suppose this still works 😉

Roman (https://rkennke.wordpress.com/2008/02/29/debugging-native-code-with-hotspot-and-gdb/)（谢谢！） 在 2008 年给我展示这个技巧！老实说，我最近没有测试过，但我想这仍然有效 😉

2. You want to start a debugging session yourself rather than automatically on crash.

2. 您希望自己启动调试会话，而不是在崩溃时自动启动。

The trick here is to either start the application in debug mode via Eclipse/Whatever or attaching the Java debugger (including jdb if you enjoy suffering!) remotely:

这里的诀窍是通过 Eclipse/Whatever 以调试模式启动应用程序，或者远程附加 Java 调试器（如果你喜欢的话，包括 jdb！

```
$ java -Dsun.awt.disablegrab=true \
       -Xdebug \
       -Xrunjdwp:transport=dt_socket,server=y,address=1080 \
       MyApplication
```

This will produce an output like the following:

这将产生如下所示的输出：

```
Listening for transport dt_socket at address: 1080
```

Blocking the application until the debugger is attached.

在附加调试器之前阻止应用程序。

At this point, you can set the breakpoints in your IDE and attach to the Java process remotely. The idea is to set the breakpoint right before the native call (*tip*: If you follow from there stepping with the java debugger, you'll also see how native libraries are loaded).

此时，您可以在 IDE 中设置断点并远程附加到 Java 进程。这个想法是在本机调用之前设置断点（*提示*：如果您从那里开始执行 java 调试器的步骤，您还将看到本机库是如何加载的）。

Now to connect gdb all you need to to is to get the *pid* of the java process, with jps for example:

现在要连接 gdb，您需要做的就是获取 java 进程的 *pid*，例如 jps：

```
$ jps
30481 Jps
27162 MyApplication <------
```

And then: 然后：

```
$ gdb -p 27162
```

Set your breakpoint in the native function of choice. Remember the name mangling, so you need to look up how the methods are actually called in native code, the naming convention is:

在所选的本机函数中设置断点。请记住名称修饰，因此您需要查找在本地代码中实际调用方法的方式，命名约定是：

```
Java_{package_and_classname}_{function_name}(JNI arguments)
```

But you need to double check exactly everything since there may be method overloads that dictate a slightly different convention.

但是你需要仔细检查所有内容，因为可能存在方法重载，这些重载决定了略有不同的约定。

If instead of using gdb from the command line you want to use your IDE the rule to follow is the same really. Afaik both Eclipse and NetBeans allow their native debugger plugins to attach to a process.

如果您想使用 IDE，而不是从命令行使用 gdb，则要遵循的规则实际上是相同的。Afaik Eclipse 和 NetBeans 都允许将其本机调试器插件附加到进程。

All that is needed now is to set your gdb breakpoints and issue a continue in the gdb shell in order to resume the Java process so that it can then hit the breakpoint you just set. From there, stepping in Java code until you enter the native function will *magically* continue the stepping inside the native function! If you use Eclipse to do both debugging this is even extremely cool since it's just like following the program inside the same editor!

现在需要做的就是设置 gdb 断点并在 gdb shell 中发出 continue，以便恢复 Java 进程，以便它可以命中您刚刚设置的断点。从那里开始，单步执行 Java 代码直到您进入本机函数，这将*神奇地*继续在本机函数内部进行单步执行！如果您使用 Eclipse 进行这两种调试，这甚至非常酷，因为它就像在同一个编辑器中跟踪程序一样！

There's one last thing to remember (other than possibly the need to set the source location in gdb or installing the OpenJDK debuginfo package for your distribution).

最后一件事需要记住（除了可能需要在 gdb 中设置源位置或为您的发行版安装 OpenJDK debuginfo 包之外）。

Hotspot uses segfaults for a number of interesting things, like deoptimise, NullPointerException etc.. Apparently, this is faster than doing specific checks and jumping around the code. This is a problem for gdb though, since it will stop every now and then to some random routines you don't really (usually!) care about:

Hotspot 将段错误用于许多有趣的事情，例如 deoptimise、NullPointerException 等。显然，这比执行特定检查和在代码周围跳转要快。不过，这对 gdb 来说是个问题，因为它会时不时地停止到一些你并不真正（通常）关心的随机例程：

```
(gdb) cont
Continuing.
Program received signal SIGSEGV, Segmentation fault.
```

Irritating, since those are all *legitimate* segfaults.

令人恼火，因为这些都是*合法*的段错误。

To avoid that just do the following in the gdb console (or from the IDE in whatever way this is handled there):

为避免这种情况，只需在 gdb 控制台中执行以下操作（或从 IDE 中以任何方式处理）：

```
(gdb) handle SIGSEGV nostop noprint pass
```

Now all the interesting work can be done without interruptions 😉

现在所有有趣的工作都可以不间断 😉 地完成

This entry was posted in Uncategorized. Bookmark the permalink.

One response to “*Debugging the JDK with GDB*”
对 “*使用 GDB 调试 JDK*” 的一个响应

1. Pingback: Debugging with Eclipse | a word in the space...
   Pingback的： 使用 Eclipse 进行调试 | 空间中的一个词...