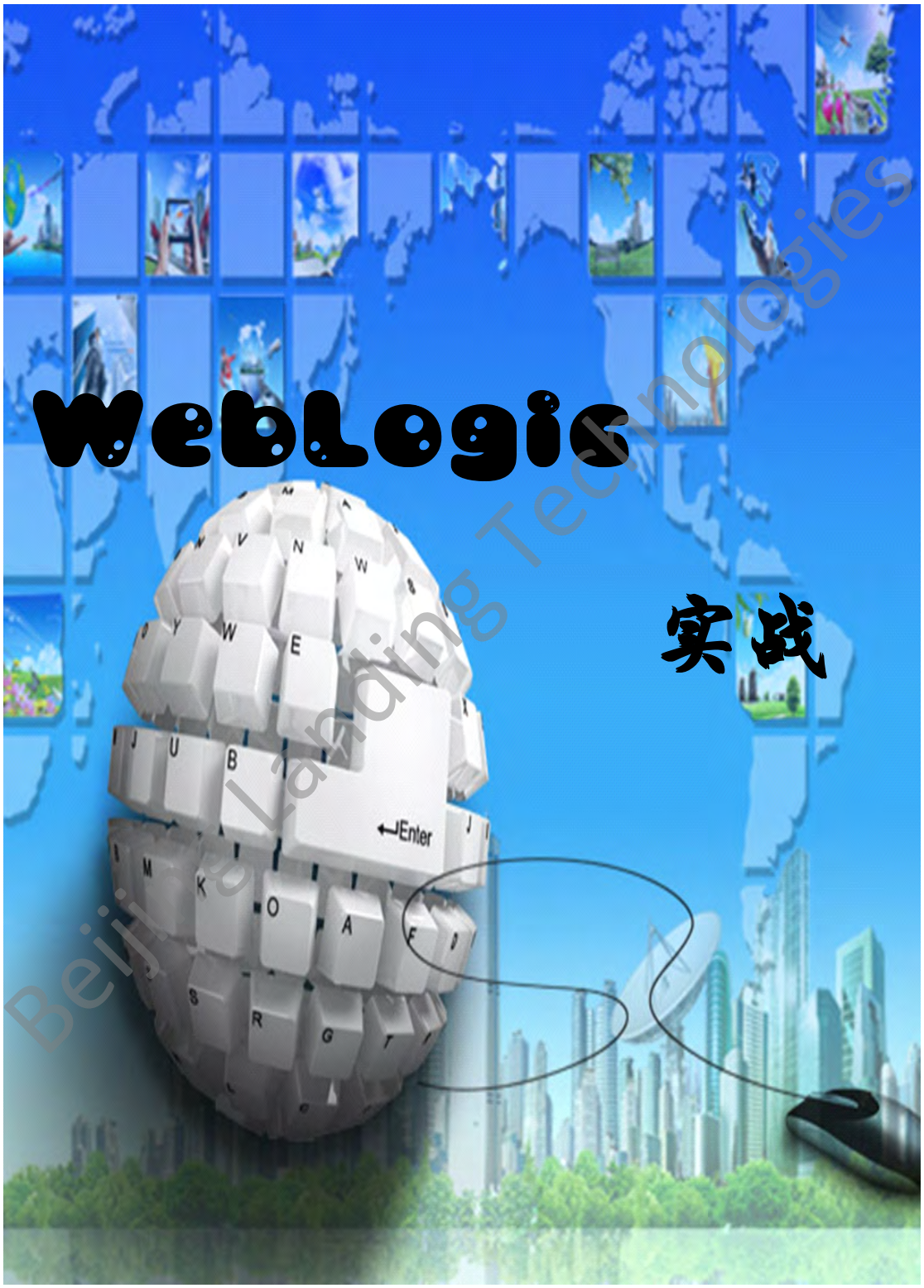
联动北方
LANDING

企业级 IT 运维宝典系列



联动北方 出品

目　录

第 15 章 服务器 core dump 分析

前面章节介绍了跟线程有关的各种状况，这下状况下，支撑 WebLogic 运行的 JVM，作为操作系统本身的进程，本身还在存续，但有时候，我们会意外的发现，WebLogic 连带其下的 JVM 突然完全从系统中消失了，而不是个别运行线程的消失，这个时候一般会发现 Java 进程退出了，而且多半会在硬盘上残留 core 文件。

15.1 什么是服务器二进制核心转储文件

如果服务器的 Java 虚拟机（Java Virtual Machine，JVM）进程异常退出，操作系统就会产生二进制核心转储（core）文件；服务器二进制核心转储文件是进程在故障点的快照转储；

- 在 Unix 系统中，会在服务器的启动目录中创建服务器二进制核心文件；
- 在 Windows 系统中，会在类似于 C:\Documents and Settings\AllUsers\Documents\DrWatson 中产生类似文件。

15.2 什么可以导致二进制核心转储文件的生成

由于 Java 代码本身是要靠虚拟机来解释执行，所以从理论上来讲，只有涉及本地操作代码的部分，才可能使服务器发生 core dump：

- 服务器崩溃；
- JVM 崩溃；
- 执行本地（C 或 C++）代码失败时可导致服务器的 JVM 失败，从而产生服务器二进制核心文件；
- 计算机崩溃；
- HotSpot 错误。

另外，要补充提醒以下相关的几点：

1、进程或文件资源限制可能会阻止产生二进制核心文件

2、二进制核心文件是探查服务器 core dump 故障原因所必需的

3、本地代码通常出现在以下位置：

- WebLogic Server 本地性能包；
- 第 2 类 JDBC 驱动程序；
- 使用 Java 本地接口(Java Native Interface，JNI）调用来访问地库的应用程序代码；
- JVM 本身。

15.3 服务器二进制转储文件探查

下面，我们来对 core 文件，展开逐步的分析。

15.3.1 探查概述

1、 确认服务器的 JVM 产生了二进制核心文件

- 一般多用操作系统自带的通用命令 file；
- 也可以用 strings 命令过滤 core 文件得到一些相关信息。

2、 使用二进制核心文件进行探查：

- 使用调试器获得堆栈跟踪；
- 确定导致故障的本地代码类型，例如，是 JDBC 驱动程序，还是应用程序代码的 JNI 调用。

3、 探查具体的故障成因

- 这项工作可能要求探查者熟谙本地代码；
- 附加的调试记录可能会有帮助；
- 排除法或替换法可能是确定真正成因的唯一手段。

4、 获得 Thread Dump

15.3.2 探查 Solaris 系统

5、 确认二进制核心文件由 JVM 产生也就是运行

```
$ file <fullpath>/core
```

示例 15-1

6、 检查 Java 开发工具包（Java Development Kit，JDK）版本

```
$ java -version                    （获得 JDK 版本）
```

示例 15-2

7、 使用调试器获得堆栈跟踪和线程信息：

- 官方技术支持部门推荐使用 dbx；
- 使用 GNU 的 gdb 也可能会获得更有用的信息。

8、 确定导致故障的本地代码类型；

9、 使用 dbx：

```
$ ls /opt/bin/dbx 或 which dbx     （获得 DBX 位置）
$ export DEBUG_PROG=/opt/bin/dbx   （设置 DBX 位置）
```

示例 15-3

启动 dbx：

```
$ dbx <path to java command>/java corefile
```

示例 15-4

输入 dbx 命令：

```
(dbx) where                        （显示堆栈摘要）
 (dbx) threads                     （显示现有线程的状态）
```

```
(dbx) quit                          (退出 dbx)
```

示例 15-5

15.3.3 探查 Linux 系统

10、确认二进制核心转储文件由 JVM 产生即运行

```
$ file <fullpath>/core
```

示例 15-6

11、检查 JDK 版本

```
$ java -version                     (获得 JDK 版本)
```

示例 15-7

12、使用 GNU 的最新版本 gdb 调试器获得堆栈跟踪和线程信息

```
$ ls /usr/local/bin/gdb             (获得 gdb 的位置)
$ export DEBUG_PROG=/usr/local/bin/gdb (设置为 gdb 的置)
```

示例 15-8

启动 gdb

```
$ gdb <path to java command>/java corefile
```

示例 15-9

13、确定导致故障的本地代码类型

输入 gbd 命令：

```
(gdb) where                         (显示堆栈摘要)
(gdb) thr                           (切换线程或显示当前线程)
(gdb) info thr                      (查询现有线程信息)
(gdb) thread apply 1 bt             (向回跟踪到 thread #1)
(gdb) quit                          (退出 gdb)
```

示例 15-10

使用 gdb bt 输出示例：

```
[root@app10 dennis]# gdb java core.11751
GNU gdb Fedora (6.8-27.el5)
Copyright (C) 2008 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later
<http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
  Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu"...
  (no debugging symbols found)
Reading symbols from /lib64/libpthread.so.0...
  (no debugging symbols found)...done.
Loaded symbols for /lib64/libpthread.so.0
Reading symbols from /usr/java/jre1.6.0_19/lib/amd64/jli/libjli.so...
  (no debugging symbols found)...done.
```

```
    Loaded symbols for /usr/java/jre1.6.0_19/bin/../lib/amd64/jli/libjli.so
    Reading symbols from /lib64/libdl.so.2...
      (no debugging symbols found)...done.
    Loaded symbols for /lib64/libdl.so.2
    Reading symbols from /usr/java/jre1.6.0_19/lib/amd64/libverify.so...
      (no debugging symbols found)...done.
    Loaded symbols for /usr/java/jre1.6.0_19/lib/amd64/libverify.so
    Reading symbols from /usr/java/jre1.6.0_19/lib/amd64/libjava.so...
      (no debugging symbols found)...done.
    Loaded symbols for /usr/java/jre1.6.0_19/lib/amd64/libjava.so
    Reading symbols from /lib64/libnsl.so.1...
      (no debugging symbols found)...done.
    Loaded symbols for /lib64/libnsl.so.1
    Reading symbols from
/usr/java/jre1.6.0_19/lib/amd64/native_threads/libhpi.so...
      (no debugging symbols found)...done.
    Loaded symbols for /usr/java/jre1.6.0_19/lib/amd64/native_threads/libhpi.so
    Reading symbols from /lib64/libnss_files.so.2...
      (no debugging symbols found)...done.
    Loaded symbols for /lib64/libnss_files.so.2
    Reading symbols from /usr/java/jre1.6.0_19/lib/amd64/libzip.so...
      (no debugging symbols found)...done.
    Loaded symbols for /usr/java/jre1.6.0_19/lib/amd64/libzip.so
    Reading symbols from /usr/java/jre1.6.0_19/lib/amd64/libnet.so...
      (no debugging symbols found)...done.
    Loaded symbols for /usr/java/jre1.6.0_19/lib/amd64/libnet.so
    Reading symbols from /usr/java/jre1.6.0_19/lib/amd64/librmi.so...
      (no debugging symbols found)...done.
    Loaded symbols for /usr/java/jre1.6.0_19/lib/amd64/librmi.so
    Reading symbols from /usr/java/jre1.6.0_19/lib/amd64/libnio.so...
      (no debugging symbols found)...done.
    Loaded symbols for /usr/java/jre1.6.0_19/lib/amd64/libnio.so
    Reading symbols from /opt/nawg/lib/libheadercodecJNI.so...
      (no debugging symbols found)...done.
    Loaded symbols for /opt/nawg/lib/libheadercodecJNI.so
    Reading symbols from /opt/nawg/lib/libwpss_wsl.so.1...done.
    Loaded symbols for /opt/nawg/lib/libwpss_wsl.so.1
    Reading symbols from /opt/nawg/lib/libwpss_hc.so.2...done.
    Loaded symbols for /opt/nawg/lib/libwpss_hc.so.2
    Reading symbols from /opt/nawg/lib/libwss_wenc.so...done.
    Loaded symbols for /opt/nawg/lib/libwss_wenc.so
    Reading symbols from /usr/java/jre1.6.0_19/lib/amd64/libawt.so...done.
    Loaded symbols for /usr/java/jre1.6.0_19/lib/amd64/libawt.so
    Reading symbols from
/usr/java/jre1.6.0_19/lib/amd64/headless/libmawt.so...done.
    Loaded symbols for /usr/java/jre1.6.0_19/lib/amd64/headless/libmawt.so
    Reading symbols from
/usr/lib/oracle/10.2.0.2/client/lib/libclntsh.so.10.1...done.
    Loaded symbols for /usr/lib/oracle/10.2.0.2/client/lib/libclntsh.so.10.1
```

```
    Reading symbols from /usr/lib/oracle/10.2.0.2/client/lib/libnnz10.so...done.
    Loaded symbols for /usr/lib/oracle/10.2.0.2/client/lib/libnnz10.so
    Reading symbols from
/usr/lib/oracle/10.2.0.2/client/lib/libociicus.so...done.
    Loaded symbols for /usr/lib/oracle/10.2.0.2/client/lib/libociicus.so
    Core was generated by `java -server -DWAPHOME=/opt/nawg/log/wps_var -
DsystemRoot=/opt/nawg -DsystemBin'.
    Program terminated with signal 6, Aborted.
    [New process 12593]
    [New process 22363]
    …
    [New process 11752]
    [New process 11751]

    #0  0x00002b08894f4215 in raise () from /lib64/libc.so.6
```

示例 15-11

然后使用执行 gdb bt 命令示例：

```
    (gdb) bt

    #0  0x00002b08894f4215 in raise () from /lib64/libc.so.6
    #1  0x00002b08894f5cc0 in abort () from /lib64/libc.so.6
    #2  0x00002b0889df13d7 in os::abort () from
/usr/java/jre1.6.0_19/lib/amd64/server/libjvm.so

    #3  0x00002b0889f2a50d in VMError::report_and_die () from
/usr/java/jre1.6.0_19/lib/amd64/server/libjvm.so
    #4  0x00002b0889df74c1 in JVM_handle_linux_signal () from
/usr/java/jre1.6.0_19/lib/amd64/server/libjvm.so
    #5  0x00002b0889df3cfe in signalHandler () from
/usr/java/jre1.6.0_19/lib/amd64/server/libjvm.so
    #6  <signal handler called>
    #7  0x00002aaaec5dd0bd in declex (yylval=<value optimized out>,
DecParam=0x572dd840) at lex_dec.c:7997
    #8  0x00002aaaec5e15c5 in decparse (DecParam=0x572dd840) at yacc_dec.c:1008
    #9  0x00002aaaec5d4ff1 in HC_DecodeHeader (Context=0x564b62f0,
WspHeader=<value optimized out>, WspHeaderLength=4096,
     ContentLength=0, HttpHeader=0x5636a390 "B$¨éB$: \r\n",
HttpHeaderLength=0x43caa814) at hc_decoder.c:2517
    #10 0x00002aaaec3b2dbe in
Java_com_nokia_wap_filter_headercodec_HeaderCodec_cDecode ()
       from /opt/nawg/lib/libheadercodecJNI.so
    #11 0x00002aaaab866058 in ?? ()
    #12 0x0000000043caa8b0 in ?? ()
    #13 0x0000000000000000 in ?? ()
    (gdb)
```

示例 15-12

从上面 bt 命令获取到的堆栈信息来看，可以清晰地看到问题所在的源代码的行数。

即：

```
hc_decoder.c:2517
```

示例 15-13

这样就便于分析和定位问题。

使用 gdb where 输出示例：

```
#6 0xfe3c6904 in __1cCosFabort6Fl_v_ () from
/wwsl/sharedInstalls/solaris/wls70sp2/jdk131_06/jre/lib/sparc/server/libjvm.so
#7 0xfe3c59f8 in __1cCosbBhandle_unexpected_exception6FpnGThread_ipCpv_v_ ()
from
/wwsl/sharedInstalls/solaris/wls70sp2/jdk131_06/jre/lib/sparc/server/libjvm.so
#8 0xfe20a8bc in JVM_handle_solaris_signal () from
/wwsl/sharedInstalls/solaris/wls70sp2/jdk131_06/jre/lib/sparc/server/libjvm.so
#9 0xff36b82c in __sighndlr () from /usr/lib/libthread.so.1
#10 <signal handler called>
#11 0xe9f90420 in Java_HelloWorld_displayHelloWorld ()from
/home/usera/wls70/solaris/projectWork/lib/libhello.so
#12 0x90aec in ?? ()
```

示例 15-14

使用 gdb thr 输出实例：

```
(gdb) thr
[Current thread is 1 (LWP 14 )]
(gdb) info thr
LWP 13 0xff29d194 in _poll () from /usr/lib/libc.so.1
LWP 12 0xff29f008 in _lwp_sema_wait () from /usr/lib/libc.so.1
LWP 11 0xff29f008 in _lwp_sema_wait () from /usr/lib/libc.so.1
LWP 10 0xff29bc2c in _so_accept () from /usr/lib/libc.so.1
LWP 9 0xff29bc2c in _so_accept () from /usr/lib/libc.so.1
LWP 8 0xff29d194 in _poll () from /usr/lib/libc.so.1
LWP 7 0xff29d194 in _poll () from /usr/lib/libc.so.1
LWP 6 0xff29f008 in _lwp_sema_wait () from /usr/lib/libc.so.1
LWP 5 0xff29f008 in _lwp_sema_wait () from /usr/lib/libc.so.1
LWP 4 0xff29f008 in _lwp_sema_wait () from /usr/lib/libc.so.1
LWP 3 0xff29d194 in _poll () from /usr/lib/libc.so.1
LWP 2 0xff29e958 in _signotifywait () from /usr/lib/libc.so.1
LWP 1 0xff29d194 in _poll () from /usr/lib/libc.so.1
LWP 16 0xff29c4fc in door_restart () from /usr/lib/libc.so.1
LWP 15 0xff369774 in private___lwp_cond_wait ()
from /usr/lib/libthread.so.1
* 1 LWP 14 0xff369764 in __sigprocmask ()
from /usr/lib/libthread.so.1
```

示例 15-15

15.3.4 探查 HP-UX 系统

14、确认二进制核心文件由 JVM 产生即运行

```
$ file <fullpath>/core
```

示例 15-16

15、检查 JDK 版本；

```
$ java -version                    （获得 JDK 版本）
```

示例 15-17

16、使用调试器获得堆栈跟踪和线程信息

- HP-UX 提供了 adb；
- 使用 GNU 的 gdb 也能获得有用的信息。

17、确定导致故障的本地代码类型

18、使用 HP-UX adb

```
$ ls /opt/bin/adb  或  which abd      （获得 adb 的位置）
$ export DEBUG_PROG=/opt/bin/adb      （设置为 adb 的位置）
```

示例 15-18

启动 dbx：

```
$ <path to java command>/java  corefile
```

示例 15-19

输入 dbx 命令：

```
adb> $C                                （显示堆栈跟踪的摘要）
adb> $r                                （显示寄存器的状态）
adb> $q                                （退出 adb）
```

示例 15-20

15.3.5 探查 AIX 系统

19、确认二进制核心文件由 JVM 产生，也就是运行

```
$ file    <fullpath>/core
```

示例 15-21

20、检查 JDK 版本；

21、使用 JVM 进程的 javacore 文件获得当前线程信息；

22、确定导致故障的本地代码类型；

23、使用 AIX javacore 文件。

javacore<WLSpid>.<ID#>.txt 文件示例：

```
    Current Thread Details:
     "ExecuteThread: '10' for queue: 'default'" (TID:0x31c70ad0,
sys_thread_t:0x3e52df68, state:R, native ID:0xf10) prio=5
    at HelloWorld.displayHelloWorld(Native Method)
    at servlets.NativeServlet.doGet(NativeServlet.java:85)
    at javax.servlet.http.HttpServlet.service
     (HttpServlet.java:740)
    at javax.servlet.http.HttpServlet.service
```

```
    (HttpServlet.java:853)
at weblogic.servlet.internal.ServletStubImpl$
ServletInvocationAction.run (ServletStubImpl.java:1058)
at weblogic.servlet.internal.ServletStubImpl.invokeServlet
(ServletStubImpl.java:401)
at weblogic.servlet.internal.ServletStubImpl.invokeServlet
(ServletStubImpl.java:306)
```

示例 15-22

15.3.6 探查 WINDOWS 系统

24、Windows 系统中的 DrWatson 日志文件 drwtsn32.log 与 Unix 二进制核心文件类似；

25、该日志文件的产生位置一般在“C:\Documents and Settings\All Users\Documents\DrWatson”

26、使用 DrWatson 探查故障原因；

27、JVM 的 hs_err_pid<WLSpid>.log 文件也可能会包含有用的信息；

28、确定导致故障的本地代码类型。

JVM 日志文件 hs_err_888.log 示例：

```
An unexpected exception has been detected in native code outside of VM
Unexpected Signal : 11 occurred at PC=0x5a4cf2e4
Function name=Java_HelloWorld_displayHelloWorld
Library=/home/spoz/wls70/linuxAS/user_projects/mydomain/lib/lib  hello.so
Current Java thread:
at HelloWorld.displayHelloWorld(Native Method)
at servlets.NativeServlet.doGet(NativeServlet.java:85)
at  javax.servlet.http.HttpServlet.service(HttpServlet.java:740)
at  javax.servlet.http.HttpServlet.service(HttpServlet.java:853)
. . .
Local Time = Wed 17 09:35:39 2004
Elapsed Time = 186
# The exception was detected in native code outside the VM
# Java VM: Java HotSpot(TM) Client VM (1.3.1_06-b01 mixed mode)
```

示例 15-23

15.3.7 未提供调试器的系统探查

如果未提供调试器，可能会提供 pstack 和 pmap 实用程序，实用程序的名称视所用平台而有所不同，您可能需要下载这些工具：

平台	类 pstack 命令	类 pmap 命令
Solaris	pstack	pmap
AIX5.2 或更高版本	procstack	procmap
Linux	lsstack	pmap
HP-UX	未提供	未提供

表 15-1

比如，使用 Solaris pstack 命令输出示例：

```
    core 'core' of 20956:/wwsl/sharedInstalls/
solaris/wls70sp2/jdk131_06/bin/../bin/sparc/native
    ----------------- lwp# 14 / thread# 25 -------------
    ff369764 __sigprocmask (ff36bf60, 0, 0, e6181d70,  ff37e000, 0) + 8
    ff35e110 _sigon (e6181d70, ff385930, 6, e6180114, e6181d70, 6) + d0
    ff361150 _thrp_kill (0, 19, 6, ff37e000, 19, ff2c0450) + f8
    ff24b900 raise (6, 0, 0, ffffffff, ff2c03bc, 4) + 40
    ff2358ec abort (ff2bc000, e6180268, 0, fffffff8, 4, e6180289) + 100
    fe3c68fc __1cCosFabort6Fl_v_ (1, fe4c8000, 1, e61802e8, 0, e9f90420) + b8
    fe3c59f0 __1cCosbBhandle_unexpected_exception6FpnGThread_ipCpv_v_ (ff2c02ac,
fe53895c, fe4dc164, fe470ab4, fe4c8000, e6180308) + 254
    fe20a8b4 JVM_handle_solaris_signal (0, 25d5b8, e6180d90, fe4c8000, b,
e6181048) + 8ec
    ff36b824 __sighndlr (b, e6181048, e6180d90, fe20a8cc, e6181e14, e6181e04) +
c
    ff3684d8 sigacthandler (b, e6181d70, 0, 0, 0, ff37e000) + 708
    --- called from signal handler with signal 11 (SIGSEGV) ---
    e9f90420 Java_HelloWorld_displayHelloWorld (25d644, e6181224, e61819b8, 0,
2, 0) + 30
    00090ae4 ???????? (e6181224, e61819b8, 25d5b8, fe4c8000, 0, 109a0)
    0008dc4c ???????? (e61812c4, ffffffff, ffffffff, 97400, 4, e61811b8)
    0008dc4c ???????? (e618135c, e61819b8, fe4c8000, 99600, c, e6181250)
    0008dc4c ???????? (e61813ec, f76a2f90, e618147c, 99600, c, e61812f8)
    0008ddb4 ???????? (e618147c, f68578b8, 0, 99974, c, e6181388)
    0008ddd8 ???????? (e618154c, e61815c8, e61815cc, 99974, 4, e6181410)
```

示例 15-24

比如，使用 pmap 命令输出示例：

```
    E9500000 1184K read
    E9680000 1392K read
    E9800000 4608K read
    E9F60000 136K read/write/exec
    E9F90000 8K read/exec
    /home/usera/wls70/solaris/project
    Work/lib/libhello.so
    E9FA0000 8K read/write/exec /home/usera/wls70/solaris/p
    rojectWork/lib/libhello.so
    E9FB4000 8K read/write/exec
    E9FC0000 120K read/exec /usr/lib/libelf.so.1
    E9FEE000 8K read/write/exec /usr/lib/libelf.so.1
    ...
```

示例 15-25

15.4 未生成二进制核心文件解决办法

如果系统并未成功生成二进制核心转储文件，我们则要仔细看是什么原因导致的，或者寻找一种有效的办法，来确保其生成。

15.4.1 确保核心转储文件的生成

确保服务器发生故障时可以产生二进制核心转储文件

- 检查系统和用户对二进制核心文件大小的限制，也就是运行“ulimit -c”；
- 在 Solaris 系统中，请使用 coreadm 命令确认没有禁用二进制核心文件转储， 也就是执行“coreadm -e process”；
- 在 Linux 系统中，缺省情况下会禁用二进制核心文件转储，也就是检查 /etc/security 中的 limits.conf 文件；
- 在 HP-UX 系统中，检查用户进程数据段大小，也就是检查 maxdsiz 值 检查磁盘空间的用户限额。

15.4.2 备用方案：获得最后时刻的 Thread Dump

Thread Dump 是对 JVM 进程中所有活动线程的原样快照，捕捉线程在发生故障前的瞬间状态。

服务器启动时启用下列项：

- Sun JVM: -XX:+ShowMessageBoxOnError
- JRockit JVM: -Djrockit.waitonerror

同时在前台启动服务器，之后如果 JVM 崩溃，会显示一则提示：

```
Do you want to debug the problem?
```

示例 15-26

回答提示前，您就可以趁机捕捉 JVM 的 Thread Dump。

15.5 故障排除检查清单

对这类问题的故障排除策略如下：

（1）确认服务器的 JVM 产生了二进制核心文件如果没有二进制核心文件，则设置相关参数在下次故障发生时捕捉二进制核心文件；

（2）使用调试器或其它工具，对二进制核心文件进行探查；

（3）探查具体的故障原因：

- 删除或替换可疑的本地代码区域；
- 可能需要进行记录和/或详细调试；

（4）继续监视，看是否还会发生故障。