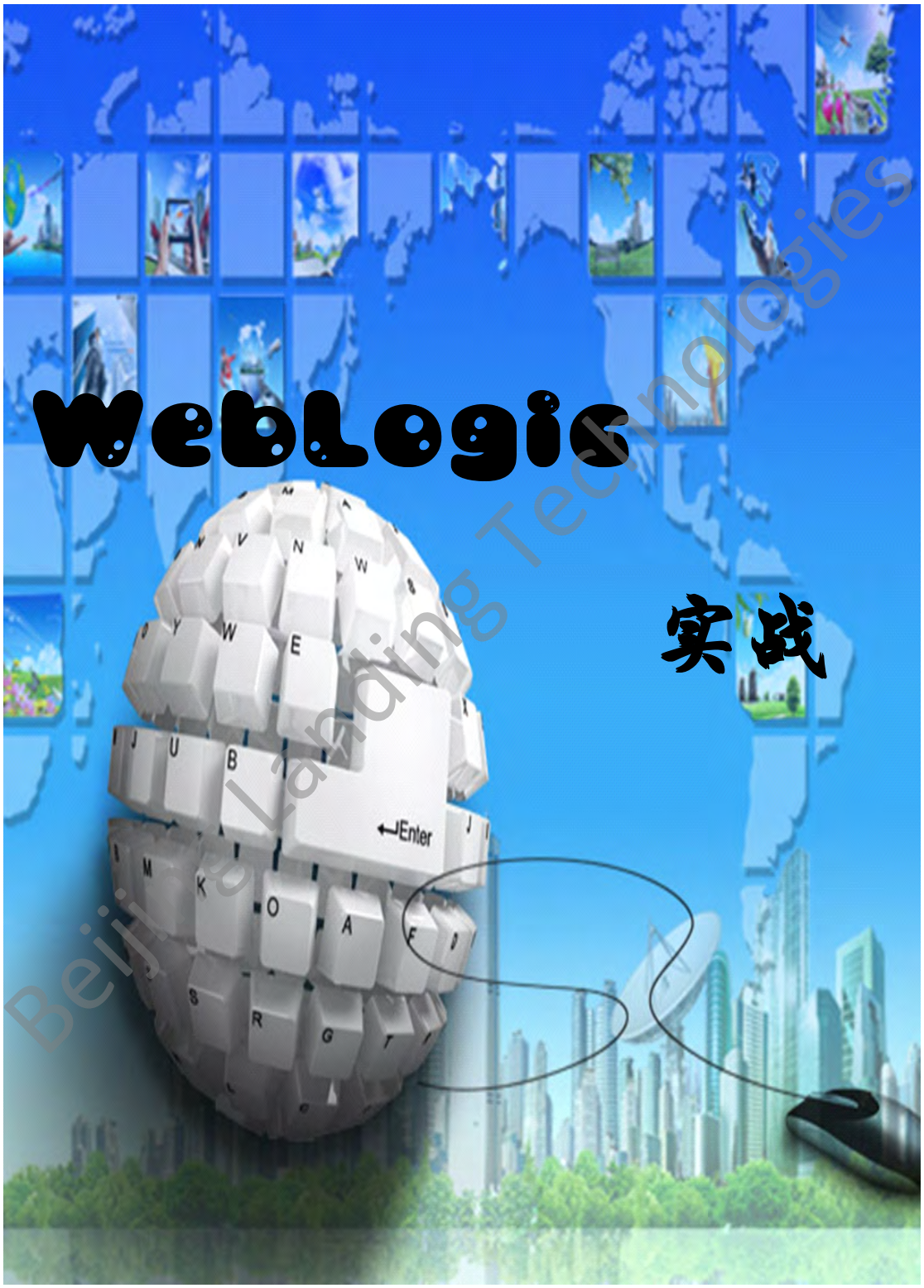
联动北方
LANDING

企业级 IT 运维宝典系列



联动北方 出品

目　录

目　录 2

第 18 章　不可恢复堆栈溢出故障 3
18.1　什么情况可导致堆栈溢出 3
18.2　堆栈溢出的故障症状 3
18.3　堆栈溢出探查 3
18.3.1　确定可以利用的信息 4
18.3.2　查看日志中的堆栈跟踪 4
18.3.3　探察二进制核心文件 4
18.3.4　探查：Solaris 系统 5
18.3.5　探查：Linux 系统 5
18.3.6　探查：HP-UX 系统 7
18.3.7　探查：AIX 系统 8
18.3.8　探查：Windows 系统 8
18.4　堆栈溢出的解决办法 9
18.5　故障排除检查清单 10

第 18 章 不可恢复堆栈溢出故障

在前面章节讲述了内存不足和内存溢出(OOM)相关内容后，作为其延伸和相关联的一种故障，不可恢复堆栈溢出故障，进入我们讨论的视野。

18.1 什么情况可导致堆栈溢出

应用程序递归过深时会发生堆栈溢出，应用程序可能会发生一般 StackOverFlow 异常。 不可恢复 的堆栈溢出是指应用程序得到"an irrecoverable stack overflow has occurred. Unexpected Signal 11" (SEGV) 消息时发生的堆栈溢出。

堆栈溢出示例输出：

```
Unexpected Signal : 11 occurred at PC=0xfb9c22ec
Function name=write (compiled Java code)
Library=(N/A)
Current Java thread:
Dynamic libraries:
0x10000 /opt/bea/jdk131/jre/bin/../bin/sparc/native_threads/java
0xff350000 /usr/lib/lwp/libthread.so.1. . .
# HotSpot Virtual Machine Error : 11
# Error ID : 4F530E43505002BD 01
#
# Please report this error at
# http://java.sun.com/cgi-bin/bugreport.cgi
# Java VM: Java HotSpot(TM) Client VM (1.3.1_07-b02 mixed mode)
```

示例 18-1

如果递归程度超过线程的堆栈大小，递归方法调用就会引发堆栈溢出。下列项目中存在递归错误可引发堆栈溢出：

- 应用代码；
- 应用服务器代码；
- JVM 自身的代码。

另外某些已知 JVM 错误也可能会导致堆栈溢出。

18.2 堆栈溢出的故障症状

日志或控制台通常会报告 StackOverflowError，不可恢复的堆栈溢出会使 JVM 崩溃，并会试图产生二进制核心文件。

通常情况下：

- 二进制核心文件可以说明，同一应用程序代码函数被反复多次调用；
- 但它不一定能准确定位到哪个具体的应用代码函数。

18.3 堆栈溢出探查

下面我们对堆栈溢出情况，展开逐步探查。

18.3.1 确定可以利用的信息

- 查看日志中的堆栈跟踪；
- 如果产生了二进制核心文件则探查该文件；
- 收集 thread dump 信息。探查具体的故障原因；
- 这项工作可能要求探查者熟谙应用程序代码；
- 应用程序调试记录可能会有帮助；
- 是否为已知 JVM 问题？

18.3.2 查看日志中的堆栈跟踪

StackOverflowError 可能会在服务器日志或控制台中显示程序调用堆栈的反向跟踪。遗憾的是，很少有提供堆栈反向跟踪的情况，因此需要采用其它诊断方法。可资利用的是，因发生不可恢复的堆栈溢出而崩溃的 JVM 可能会产生 JVM 日志文件，其中包含有可能导致了该二进制核心文件的库的详细信息。

JVM 的日志文件：

- 位于服务器的启动目录中；
- 通常名为 hs_err_pid<WLSpid>.log ，其中 <WLSpid> 是服务器进程的进程 ID。

JVM.log 实例文件：

```
An unexpected exception has been detected in native code outside the VM.
Unexpected Signal : 11 occurred at PC=0x5a4cf2e4
Function name=Java_HelloWorld_displayHelloWorld
Library=/home/spoz/wls70/linuxAS/user_projects/mydomain/lib/libhello.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 18-2

18.3.3 探察二进制核心文件

如果 StackOverflowError 导致 JVM 崩溃，通常情况下会产生二进制核心文件。可以查看二进制核心文件，了解 JVM 终止的那一刻发生的情况。但在某些情况下，也可能不会产生二进制核心文件：

- WebLogic 可能会正常关闭，因而不会产生二进制核心文件；
- 如果 JVM 不崩溃，则不会产生二进制核心文件；

- 进程或文件大小限制可能会阻止产生二进制核心文件。

可以采取措施来确保能够产生二进制核心文件。二进制核心文件往往是探查服务器故障原因所必需的，要确保服务器发生故障时可以产生了二进制核心文件：

- 检查二进制核心文件的系统和用户大小限制，也就是运行“ulimit -c”；
- 在 Solaris 系统中，检查 /etc/system 中的二进制核心文件转储大小设置，也就是检查“set sys:coredumpsize=0”；
- 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；
- 使用“quota -v”命令检查磁盘空间的用户限额。

下面，我们来对各个操作系统环境逐一进行讲述，为节省篇幅计，UNIX/类 UNIX 系统重点讲述 Linux 环境，其余的系统基本类似，只是个大致的概览。

18.3.4 探查：Solaris 系统

在 Solaris 上，可以使用 Solaris dbx 工具来探查。

确认 JDK 版本：

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

示例 18-3

启动 dbx 工具：

```
$ ls /opt/bin/dbx    或    which dbx   （获得 dbx 的位置）
$ export DEBUG_PROG=/opt/bin/dbx   （设置为 dbx 的位置）
$ dbx <path to java command>/java  corefile
```

示例 18-4

输入 dbx 子命令来寻找当前堆栈和线程：

```
(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 18-5

18.3.5 探查：Linux 系统

确认二进制核心文件由 JVM 产生，也就是运行

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

示例 18-6

确认 JDK 版本：

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

示例 18-7

使用最新版本 GNU 的 gdb 调试器获得堆栈跟踪和线程信息，确定引发故障的方法，使用 GNU gdb。

启动 gdb:

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

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

示例 18-8

输入 gdb 子命令：

```
(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                    （反向跟踪到 thread #1）
(gdb) quit                                 （退出 gdb）
```

示例 18-9

使用 gdb where 的示例：

```
gdb) where
#0 0xff369764 in __sigprocmask () from /usr/lib/libthread.so.1
#1 0xff35e978 in _resetsig () from /usr/lib/libthread.so.1
#2 0xff35e118 in _sigon () from /usr/lib/libthread.so.1
#3 0xff361158 in _thrp_kill () from /usr/lib/libthread.so.1
#4 0xff24b908 in raise () from /usr/lib/libc.so.1
#5 0xff2358f4 in abort () from /usr/lib/libc.so.1
#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/shared
Installs/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 18-10

使用 gdb thr 的示例：

```
(gdb) thr
[Current thread is 1 (LWP 14 )]
(gdb) info thr
16 LWP 13 0xff29d194 in _poll () from /usr/lib/libc.so.1
15 LWP 12 0xff29f008 in _lwp_sema_wait () from /usr/lib/libc.so.1
14 LWP 11 0xff29f008 in _lwp_sema_wait () from /usr/lib/libc.so.1
13 LWP 10 0xff29bc2c in _so_accept () from /usr/lib/libc.so.1
12 LWP 9 0xff29bc2c in _so_accept () from /usr/lib/libc.so.1
11 LWP 8 0xff29d194 in _poll () from /usr/lib/libc.so.1
10 LWP 7 0xff29d194 in _poll () from /usr/lib/libc.so.1
```

```
9 LWP 6 0xff29f008 in _lwp_sema_wait () from /usr/lib/libc.so.1
8 LWP 5 0xff29f008 in _lwp_sema_wait () from /usr/lib/libc.so.1
7 LWP 4 0xff29f008 in _lwp_sema_wait () from /usr/lib/libc.so.1
6 LWP 3 0xff29d194 in _poll () from /usr/lib/libc.so.1
5 LWP 2 0xff29e958 in _signotifywait () from /usr/lib/libc.so.1
4 LWP 1 0xff29d194 in _poll () from /usr/lib/libc.so.1
3 LWP 16 0xff29c4fc in door_restart () from /usr/lib/libc.so.1
2 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 18-11

使用 gdb bt 的示例：

```
(gdb) thread apply 1 bt
Thread 1 (LWP 14 ):
#0 0xff369764 in __sigprocmask () from /usr/lib/libthread.so.1
#1 0xff35e978 in _resetsig () from /usr/lib/libthread.so.1
#2 0xff35e118 in _sigon () from /usr/lib/libthread.so.1
#3 0xff361158 in _thrp_kill () from /usr/lib/libthread.so.1
#4 0xff24b908 in raise () from /usr/lib/libc.so.1
#5 0xff2358f4 in abort () from /usr/lib/libc.so.1
#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 ?? ()
#13 0x8dc54 in ?? ()
. . .
```

示例 18-12

18.3.6 探查：HP-UX 系统

确认二进制核心文件由 JVM 产生，也就是运行：

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

示例 18-13

在 HP-UX 上确认 JDK 版本：

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

示例 18-14

使用调试器获得堆栈跟踪和线程信息：

- HP-UX 默认提供了 adb；
- 使用下载的 GNU 的 gdb 也可以获得有用的信息。

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

示例 18-15

输入 adb 子命令：

```
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 18-16

18.3.7 探查：AIX 系统

确认二进制核心文件由 JVM 产生，也就是运行：

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

示例 18-17

检查 JDK 版本，并使用 JVM 进程的 javacore 文件获得当前线程信息，确定引发故障的方法。

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 18-18

18.3.8 探查：Windows 系统

Windows 系统中的 DrWatson 日志文 drwtsn32.log 与 Unix 二进制核心文件类似，该日志文件位置一般在 C:\Documents and Settings\All Users\Documents\DrWatson。

JVM 的 hs_err_pid<WLSpid>.log 文件（如果有）也可能会包含有用的信息，用以确定引发故障的方法。

如果未提供调试器，各系统可能会提供 pstack 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	相关 pstack 工具
Solaris	pstack
AIX5.2 或更高的版本	procstack
Linux	Lsstack
HP-UX	未提供

表 18-1

18.4 堆栈溢出的解决办法

首先按照上述探查方法，确定堆栈溢出问题的成因，然后您可能需要采取以下措施：

1. 修改 JVM 参数增加 JVM 的线程堆栈大小：

每个 Java 线程有两个堆栈，一个用于 Java 代码，一个用于 C 代码 JVM 的参数。 -Xss 控制 Java 代码的堆栈大小，如果错误的递归调用是问题所在，可以考虑适当调高-Xss 的值，不过这种方法通常并不能避免堆栈溢出，而只能延缓它的发生。

2. 纠正应用程序代码中的错误

确保递归算法不会导致死循环并可以返回结果，如果怀疑应用程序代码有问题，利用以下代码片段可能有助于查明问题所在：

```
catch (StackOverflowError e)
{
    System.err.println("Exception: " + e );
    // Here is the important thing to do
    // when catching StackOverflowErrors:
    e.printStackTrace();
    // do some cleanup and destroy the thread or unravel if possible.
}
```

示例 18-19

查看最近对应用程序代码所做的所有更改，看其中是否有递归调用。在有可疑代码的地方添加调试语句。

比如，jsp_error 页中有以下标记将导致无限递归：“<%@ page errorPage="jsp_error"%>”。如果错误处理代码内出现异常，请删除此标记并输出堆栈跟

踪。这样可以在错误页面中找到问题。解决此问题之后，就可以查看发送到此页面上的原始错误。

WebLogic JSP 表单验证标记也可能导致递归错误。确保<wl:form> 的确未将 action 属性设置为包含该<wl:form>的页面，这样会导致无限循环，结果将导致堆栈溢出错误。

3. 检查已知 JVM 问题。

使用对象数组的数组时，可能会发生堆栈溢出错误。已知在 java.util 包（特别是 JVM v1.3.1 和 1.4.1）中使用 getProperty() 时，也可能会发生堆栈溢出错误。

比如，不要执行以下代码行：

```
Properties p = new Properties(System.getProperties());
```

示例 18-20

而作为替代，执行：

```
Properties p = new Properties();
p = System.getProperties();
```

示例 18-21

18.5 故障排除检查清单

对这类问题的故障排除策略如下：

（1）探查日志文件，了解出错位置，查看崩溃的服务器产生的二进制核心转储文件；

（2）如果没有二进制核心转储文件，则设置相关参数，在下次故障发生时捕捉二进制核心文件；

（3）进行 Thread Dump，找到递归方法调用；

（4）向可疑应用程序代码添加调试跟踪；

（5）检查应用程序基础结构代码：

- jsp_error 页，login/auth/error 部分代码
- WebLogic JSP 表单验证标记

（6）检查已知 JVM 问题。