Wissenschaftliches Schreiben WS 2008/09

Autor: Gamif110
HfT-Stuttgart

02. November 2008
Verbesserte Version: 05. November 2008
Weitere Veränderungen: 12. November 2008

Flight-Recorder

eine Applikation, die Speicherabbilder nach Objekte analysiert und auf deren Elemente zugreift.

Inhaltsverzeichnis

1 Einleitung

1.1 Einleitung und Umfeld des Themas

Ein Flight-Recorder stellt ein Programm dar, welches zur Aufgabe die Analyse von Speicherabbildern und dessen Aufzeichnung hat.

Des Weiteren dient es der Unterstützung der Fehlersuche. Es soll Speicherabbilder von Anwendungen nach unkontrollierten, nicht mit Regelmäßigkeit auftretenden Fehlern, untersuchen. Man versucht somit, einfach und schnell an Informationen zur Behebung von Fehlerfällen zu kommen.
Nicht jede Fehlermeldung ist ohne weitere Informationen verständlich und nachvollziehbar. Der Flight-Recorder verbessert die Nachvollziehbarkeit von Fehlerfällen, Abstürzen oder hängenden Programmen, indem man das letzte mögliche Speicherabbild analysiert und Variablen und Objekte nach ihren Inhalten prüft. Das Programm durchläuft Addressräume, Laufzeitumgebungen und Speicher, bis es auf die Ebene von Objekten gelangt. Ausgewählte Objekte werden nach Variablen und Unterobjekten ausgewertet. Die Aufgabe des Programms ist, Fehlerfälle besser verständlich zu machen und genauer entgegen zu gehen.

Zur Fehlersuche kann auch der Debug-Modus einer Entwicklungsumgebung verwendet werden. Dieser beschränkt sich auf einfache Fehler, z. B. falscher Variablentyp beim Abspeichern eines Wertes, die sich an eine Regelmäßigkeit halten. Diese Vorgehensweise bietet nicht die Möglichkeit der Informationswiedergabe von Daten der Umgebung, in der das Programm, zum Zeitpunkt des Absturzes, lief.

Es stellt sich die Frage, ob es möglich ist, mit dem Diagnostic Toolkit and Framework for Java (DTFJ) [1] ein Programm zu erstellen, das aus Speicherabbildern Daten und Objekte extrahieren kann und dies anschließend auf z/OS zu portieren.

1.2. Begriffserklärungen

Debug-Modus/ Debugger::
Teil einer Entwicklungsumgebung/ Software-Werkzeug, zur Diagnostik von Programmabläufen, Auffinden und Beheben von Fehlern

DTFJ:
Diagnostic Toolkit and Framework for Java
beschriebt eine Anzahl von Klassen, die mithilfe von Reflection die Möglickeit ergeben, Speicherabbilder von Programmen auszulesen und zu analysieren.

Reflection:
Unter Reflection versteht man das dynamische Nachladen und Verändern von Klassen. Es kann zur Laufzeit Veränderung an den Klassen vorgenommen werden.

Java:
objektorientierte Programmiersprache, die, durch ihre Einfachheit, zu großer Beliebtheit gelangte

JVM – Java Virtual Machine:
Laufzeitumgebung, in der Java-Programme ausgeführt werden.
Java-Programme laufen nicht direkt im Betriebssystem sondern in einer eigenen Umgebung, ohne plattformspezifischen Maschinencode. So wird gewährleistet, daß Programme für verschiedene Plattformen nicht neu kompiliert werden müssen.

z/OS:
Großrechner-Betriebssystem der Firma IBM

Trace:
Traces dienen der Spurenverfolgung der Fehler. Es werden Anhaltspunkte zum Ort des Fehlers ausgegeben.

Buffer:
Zwischenspeicher zur temporären Haltung von Daten, bis Ressourcen wieder frei sind.

2 Speicherabbilder

2.1. Was ist ein Speicherabbild?

Ein Speicherabbild ist eine Kopie der Informationen, die im Arbeitsspeicher zu finden sind. Aus der englischen Literatur gibt es für Speicherabbild andere Ausdrücke. Diese sind Image, Dump oder Imagedump. Ein Unterschied zu einer einfachen Kopie ist, daß nicht einfach nur die Daten und Dateien kopiert werden. Es werden auch Verwaltungsdaten gespeichert. Die Struktur des Datenträgers wird abgebildet.

2.2. Typen von Speicherabbilder

Folgende Typen von Speicherabbildern können erzeugt werden [2]:

- Console dump
- System dump
- Heap dump
- Java (core) Dump

Console dump

Ein Console dump bietet eine sehr beschränkte Möglichkeit der Analyse.
Er gibt nur Einsicht in den jeweiligen Status in Java Threads.
Geschrieben werden die Informationen über den Stderr-Kanal.

Heap dump

Ein Heapdump generiert ein Abbild aller bei Erstellung lebenden Java-Objekte, die sich im Java Heap befanden. Diese Art von Speicherabbild erzeugt eine binär komprimierte Datei. Heapdumps werden im phd-Format erzeugt.

Java dump

ein Javadump spiegelt eine interne Analyse der Java Virtual Machine wieder.
Hieraus kann man Informationen über Java Threads, geladene Klassen und Statistiken üder die Heaps bekommen. Dateien, die über Javadump generiert werden, enthalten Diagnoseinformationen über die Java Virtual Machine sowie das Programm, während der Ausführung. Die Informationen können das Betriebssystem, die Umgebung des laufenden Programms, Speicherbereiche oder Verklemmungen betreffen.
Ein anderer Name für Javadump ist Javacore. Java dump und System dump sind verschiede Abbilder.

System dump

System-Speicherabbilder nehmen den größten Speicherplatz ein, da sie den kompletten Addressraum wiederspiegeln. Auf einer z-Architektur kann ein sich ein Programm über mehrere Adressräume erstrecken. Diese Rechner haben sehr große Speicher. Es kann somit passieren, daß das Speicherabbild mehrere Gigabyte groß ist.
Auf Windowsarchitekturen sind die Adressräume pro Prozess auf 1 begrenzt.

2.3. Generierung von Dumps

Es stehen mehrere Möglichkeiten zur Verfügung, einen Dump zu generieren.
Man kann per Aufruf eines Javaprogramms Optionen angeben, wie und wann ein Speicherabbild erstellt werden soll. Eine weitere Option ist es, innerhalb des Javaquellcodes eine Anweisung zu notieren, die das System veranlaßt, den momentanen Zustand des Programms auf die Festplatte zu schreiben. Ein drittes Verfahren funktioniert über das Betriebssystem und ist nicht immer möglich. Man kann das Betriebssystem per Anweisung veranlassen, einen Systemdump zu produzieren.

Dump durch Programmaufruf

Beim Aufruf eines Programmes von der Konsole oder einem Befehlsinterpreter gibt es mehrere Optionen und Parameter, die man einem Programm übergeben kann. Die Java Virtual Machine kann man auf gleiche Weise über den Aufruf eines Programms steuern. Mit der Option -Xdump wird angegeben, zu welchem Zeitpunkt welche Art von Speicherabbild erzeugt werden kann [3].

Die erste Angabe nach -Xdump: stellt die Art von Dump dar.
Die Optionen sind:

- tool: startet externe Anwendungen, eine Art Konsole
- Console: allgemeine Angaben über Threads, die auf den Kanal stderr geschrieben werden.
- snap: Es werden Traces auf den Buffer geschrieben.
- heap: Heapdump, der alle lebende Java Objekte enthält
- java: erstellt eine Zusammenfassung über die Anwendung
- system: erstellt ein komplettes Systemdump in Binärformat

Die nächste Option in der Anweisung stellt die Events (Ereignisse) dar, wann ein Dump ausgelöst wird. Folgende Auflistung stellt eine Auswahl dar.

- gpf = ein genereller Fehler ist aufgetreten
- user = die JVM empfang ein SIGOUT/ SIGQUIT-Signal vom Betriebssystem
- abort = die JVM empfang ein SIGABRT-Signal
- vmstart = die JVM wurde gestartet
- vmstop = die JVM wurde gestoppt
- catch = eine Java-Exception wurde abgefangen
- uncaught = eine Java-Exception wurde nicht abgefangen
- thrstart = ein neuer Thread wurde gestartet
- thrstop = ein Thread wurde gestoppt

Ein Beispiel: java -Xdump:system:events=user

Dump durch Java-Anweisung

Die Firma IBM hat eine eigene Version der Java Virtual Machine, die sie für ihre Software-Entwicklungen benutzt. Im Package com.ibm.jvm.Dump sind folgende Funktionen/ Methoden untergebracht, die einen Dump auslösen:

- com.ibm.jvm.Dump.SystemDump()
- com.ibm.jvm.Dump.HeapDump()
- com.ibm.jvm.Dump.JavaDump()

Diese Funktionen lassen sich aus einem Java-Programm heraus starten.
Der generierte Status ist der, der zur Zeit der Anweisungsausführung vorgefunden wird.

Für die Entwicklung des Flight-Recorders wurde alleine die Anweisung com.jbm.jvm.Dump.SystemDump() verwendet.

Dump durch Betriebssystem-Anweisung/ Signal

Es gibt Situationen, in denen ein Programm nicht mehr reagiert und es intern nicht mehr die Möglichkeit gibt, ein Speicherabbild zu erstellen. In solchen Situationen muß man auf Betriebssystem-Ebene das Erstellen eines Speicherabbilds starten. Hierfür stehen betriebssystembedingte Anweisungen zur Verfügung.

Speicherabbild durch JEXTRACT mit Index versehen

Um die System-Speicherabbilder unabhängig der Maschine von der sie kommen, bearbeiten zu können, müssen sie mit einem Index versehen werden. System-Speicherabbilder verschiedener Systeme haben jeweils ein eigenes Format und können ohne weitere Hilfe nicht vom DTFJ gelesen werden. Verschiedene Wortgröße, Endians und Arten von Datenstrukturen erschweren es, ohne Vorbereitungen Informationen verschiedener Systeme bereitzustellen. Um diese Hürde zu nehmen und die Informationen der Dumps bereitzustellen und auszuwerten, müssen sie, bevor man mit der DTFJ-API darauf zugreifen kann, mit JEXTRACT vorbereitet werden. JEXTRACT erstellt eine XML-Datei, die den Dump beschreibt. Hiermit ist es später möglich die Lage der Datenstrukturen innerhalb des Dumps zu identifizieren. Nachdem die Vorbereitungen durch Dumperstellung und anschließender Formatierung durch JEXTRACT vollendet sind, kann man durch die DTFJ-API auf den Speicher des zu untersuchenden Programms zugreifen [4].

3 Erstellung des Flight-Recorders

3.1. Vorgehensweise mit DTFJ

Das Diagnostic Toolkit and Framework for Java liefert Möglichkeiten, um auf Adressräume, Laufzeitumgebungen, Heap, Klassen und deren Objekte und Variablen zuzugreifen und auszulesen.

Die Technik basiert auf Reflection und gibt die Möglichkeit über verschachtelte while-Schleifen über mehrere Ebenen von Adressräumen, Runtimes und Heap zu Objekten zu gelangen und diese abzufragen.

Felder von Objekten werden über deren Definition innerhalb der Klasse abgefragt.

Ein Programm, das untersucht werden soll, kann von mehreren Adressräumen, Laufzeitumgebungen, Heaps sowie Klassen und Objekte abhängen. Zu diesem Zweck steht eine Technik zur Verfügung, die auf Iteratoren und while-Schleifen beruht. Auf jeder Ebene steht ein Iterator zur Verfügung, der alle Objekte der Ebene bedient und den Zugriff ermöglicht.

3.2. Programmierung auf Windows-Architektur

Im Folgenden soll das Szenario beschrieben werden, wie man auf Objekte/ Instanzen von Klassen zugreift und deren Variablen ausliest. Genau dies ist die Aufgabe des Flight-Rekorders, Daten und Objekte von Java-Programmen darzustellen.

Um auf Objektebene zuzugreifen, ist es nötig, verschiedene höher stehende Ebenen zu durchlaufen. Man muß Informationen zu Adressräume, Java-Laufzeitumgebungen und Heap durchlaufen, um an JavaObjekte zu gelangen.

Der erste Schritt ist, dem DTFJ den Zugriff auf das komprimierte Verzeichnis mit dem Speicherabbild und der dazugehörigen XML-Datei zu ermöglichen. Dies geschieht nach allgemeinen Javabedingungen. Hierzu lassen sich die Methoden und Konstruktoren der File-Klasse zur Erstellung von File-Objekten benutzen.

Daraufhin erstellt man mittels Reflection ein ImageFactory-Objekt. Mittels ImageFactory-Object wird das File-Objekt an ein statisch angelegtes Image-Objekt übergeben, welches den Einstieg in die Analyse des Speichers darstellt.

Nach diesen programmiertechnischen Vorarbeiten kann man auf Adressräume und deren untergeordnete Ebenen zugreifen [5].

Zugriff auf Adressräume

Auf einem Großrechner gibt es die Möglichkeit, daß einem Programm mehrere Adressräume zugeteilt werden. Somit müssen, auf der Suche nach den richtigen Informationen, alle vorhandenen Adressräume durchlaufen werden oder man gibt einen bestimmten an. Eine Möglichkeit ist, dies mit einem Iterator und einer while-Schleife durchzuführen. Man legt einen Iterator an und ruft die Methode getAddressSpaces() des Objektes Image auf. Innerhalb des Schleifenkopfes wird der Iterator darauf geprüft, ob er weitere Objekte enthällt. Im Schleifenkörper wird durch die Methode Iterator.next() ein Objekt des Adressraumes angelegt. Dieses Objekt läßt sich nach Informationen abfragen.

Ein Auszug der Informationen, die sich auf der Ebene der Adressräume darstellen lassen:

- Abgleich auf andere Objekte mittels AddressSpace.equals(object)
- Rückgabe des Prozesses innerhalb des Adressraumes, durch Iterator
- Man kann sich die unbehandelten Speicherbereiche zurückgeben lassen
- kein direkter Zugriff auf Variablen

Zugriff auf Prozesse

Jedem Adressraum sind Prozesse zugeteilt. Diese stellen die nächste zu überbrückende Ebene dar. Um Objekte abfragen zu können, werden die Prozesse abgefragt und iteriert. Dazu stellt das Framework dieselben Instrumente zur Verfügung, wie in der Ebene der Adressräume. Innerhalb der while-Schleife der Adressräume legt man mithilfe der Methode getProcesses() einen Iterator an, der den Kopf der while-Schleife darstellt. Innerhalb der while-Schleife erstellt man mit jedem Iteratordurchgang ein Objekt der Klasse ImageProcess.

Auszug der Informationen, die sich über Prozesse abfragen lassen:

- Abgleich mittels ImageProcess.equals(object)
- ImageProcess.getLibraries() liefert einen Iterator über geladenen Bibliotheken
- ImageProcess.getEnvironment() liefert die Umgebungsvariablen für diese Prozess
- ImageProcess.getSignalNumber() liefert die Betriebsystemnummer des Signals, daß die Erstellung des Speicherabbildes gestartet hat
- ImageProcess.getCurrentThread() liefert den ImageThread, der die Erstellung des Speicherabbildes ausgelöst hat
- ImageProcess.getThreads() liefert einen Iterator über die Threads, die sich im Image befinden
- kein direkter Zugriff auf Variablen

Zugriff auf die JavaRuntime

Das Objekt ImageProcess stellt einen Iterator zur Verfügung, der den Zugriff auf die Laufzeitumgebung der Prozesse zurückgibt. Die Methode hierfür heißt Imageprocess.getJavaRuntimes() und gibt einen Iterator auf die Laufzeitumgebungen wieder. Die Technik beruht auf einer while-Schleife, die den Iterator weiter zählen läßt. Innerhalb des Körpers wird ein Objekt der Klasse JavaRuntime angelegt.

Auszug der Informationen auf dieser Ebene:

- Abgleich mittels JavaRuntime.equals(object)
- JavaRuntime.getCompiledMethods() liefert einen Iterator, der alle kopilierten Methoden wiederspiegelt
- JavaRuntime.getHeaps() liefert einen Iterator zu den Heaps in der Virtual Machine
- JavaRuntime.getJavaVM() liefert ein Objekt, das die Virtual Machine representiert
- kein direkter Zugriff auf Variablen

Zugriff auf Heaps

Die Ebene der JavaRuntime ermöglicht den Zugriff auf Objekte der Klasse JavaHeap.

Folgende Informationen stehen auf dieser Ebene zur Verfügung:

- JavaHeap.getName() liefert eine kurze Beschreibung des Heaps
- JavaHeap.getSections() liefert einen Iterator zu den Speicherbereichen, die der Heap belegt.
- JavaHeap.equals(object) liefert einen Vergleich auf ein Objekt
- kein direkter Zugriff auf Variablen

Zugriff auf Objekte

Innerhalb der Ebene der JavaHeap-While-Schleife wird ein Iterator instanziert, der den Zugriff auf die Objekte des zu untersuchenden Programms ermöglicht. Um alle Objekte zu erfassen, wird die Technik des Iterators angewandt. Innerhalb einer while-Schleife wird ein Objekt der Klasse JavaObjekt angelegt. Diese spiegeln die Objekte des angegebenen Programms wieder.

Folgende Informationen und Methoden stehen zur Verfügung:

- JavaObject.getJavaClass() liefert die Klasse von der das Objekt abstammt
- JavaObject.equals() vergleicht zwei Objekte
- JavaObject.isArray() überprüft, ob ein Objekt ein Array ist
- JavaObject.arraycopy(int, object, int, int) kopiert Teile des Object-Arrays in ein anderes
- JavaObject.getID() liefert eine eindeutige Identifikation zu einem Objekt
- JavaObject.getSize() liefert die Größe eines Objektes
- kein direkter Zugriff auf Variablen

Auf Objektebene kann man nicht auf deren Namen zugreifen. Hier besitzen Objekte Identifikationsnummern. Um die Anzahl der Objekte für eine Suche zu begrenzen, wird die Methode getJavaClass() bedient, die die Klasse wiedergibt, von der das Objekt instanziert wurde.

Zugriff auf JavaClass

Der Zugriff auf Objekte der Klasse JavaClass ist wichtig, um Felder und Methoden auslesen zu können.

Auszug der Informationen auf dieser Ebene:

- JavaClass.equals(object) vergleicht zwei Klassen miteinander
- JavaClass.getComponentType() liefert die Klasse der Objekte innerhalb eines Arrays
- JavaClass.getDeclaredFields() liefert einen Iterator über die Felder einer Klasse
- JavaClass.getDeclaredMethods() liefert einen Iterator über die Methoden einer Klasse
- kein direkter Zugriff auf Variablen

Felder und Methoden eines Objektes abfragen

Für die Abfrage von Variablen und Methoden von Objekten steht direkt keine Methode innerhalb von JavaObject zur Verfügung. Abfragen über Elemente eines Objektes werden über die Klasse JavaClass realisiert.

Jedes JavaObject verfügt über eine Methode JavaObject.getJavaClass(). Diese gibt als Returnwert ein Objekt der Java-Klasse wieder, von der das Objekt instanziert wurde. Die Klasse JavaClass besitzt eine Methode, die einen Iterator zu den Feldern einer Klasse zurückgibt. Diese heißt: JavaClass.getJavaFields(). Mit diesem Iterator kann in einer weiteren Schleife jedes Feld einer beliebigen Klasse ausgegeben werden. Für Methoden eines Objekts funktioniert es gleich, jedoch mit der Methode JavaClass.getJavaMethods().

Auszug der Informationen der Klasse JavaField:

mittels eines Objectes der Klasse JavaField lassen sich die Informationen aus Feldern, elementaren Datentypen, auslesen. Für jeden elementaren Datentyp gibt es eine Funktion get..., getInt(object), getDouble(object), getChar(object), etc.

weitere Methoden:

- JavaField.getModifier() liefert einen Int-Wert, der Modifizierer, wie private, public, transient, etc. zurückgibt.
- JavaField.getName() liefert den Namen der Variable zurück
- JavaField.getSignature() liefert den Typ einer Variable
- kein direkter Zugriff auf Variablen

4 Portierung auf z/OS

In Java geschriebene Programme sind betriebssystemunhabhängige Applikationen, die in einer eigenen Umgebung, der JVM – Java Virtual Machine, laufen. Somit müssen sie nicht für jedes Betriebssystem übersetzt werden.

Trotzdem können Probleme beim Portieren auf andere Systeme entstehen.

4.1. Problematiken und aufgetretene Probleme

Größe des Temp-Speichers

Bei der Vorbereitung des Speicherabbildes wird ein komprimiertes Verzeichnis mit Inhalt des Dump-Files und der dazugehörigen Beschreibungsdatei in XML-Format erstellt. Dieses wird dem Flight-Recorder beim Programmstart übergeben, der den Inhalt extrahiert. In Abhängigkeit der Größe des Speicherabbildes, daß auf z/OS mehrere hundert MB groß sein kann, muß ein temporärer Ordner existieren, der der Größe des extrahierten Speicherabbildes und der XML-Datei genügt. Im Fehlerfall wird auf der Konsole eine Nachricht ausgegeben, die auf einen zu kleinen Speicher hinweist. In dem Fall hilft eine Vergrößerung des Temporären Ordners oder die Angabe eines Ordners mit größerem Volumen.

Übereinstimmende Java-Version

Die Java-Version des zu analysierenden Anwendung und der der JVM, unter der JEXTRACT ausgeführt wird, sollte dieselbe sein. Es wird empfohlen, daß das Ausführen des Dumps und die Vorbereitung auf derselben Maschine abgearbeitet werden. Eine mögliche Fehlermeldung beim Starten des Flight-Recorders ist, daß der Dump und die Version von JEXTRACT nicht miteinander kompatibel sind. Konsolenausgabe: „J9RAS.length is incorrect. ... This Version of JEXTRACT is incompatible with this dump“.

Größe des Speicherabbildes

Die Größe eines System-Speicherabbildes auf einem Großrechner, kann, bedingt durch die Benutzung mehrerer Adressräume, höherer Funktionalität und der Größe des Arbeitsspeichers gesamt größer sein als die auf einem Windows-Desktop-System. Eine Größe von über 100 Megabyte kann erreicht werden. Die Laufzeit der Analyse und eine Ausgabe der Daten kann größer ausfallen, als auf Windows-Desktop-Rechnern.

5 Zusammenfassung, Fazit und Ausblick:

Zunächst muß ein System-Dump nach einem Programmfehler erstellt werden. Dieser sollte mit einem Tool vorbereitet werden, damit systemspezifische Informationen allgemein zugänglich gemacht werden können. Im Anschluß kann mit einem Java-Programm, dass mit dem DFTJ-Framework erstellt worden ist, dieser Dump analysiert werden und die Ergebnisse in eine Datei gespeichert oder auf der Konsole ausgegeben werden.

Resümierend lässt sich feststellen, dass ein Flight-Recorder eine komfortable und funktionierende Möglichkeit darstellt, auf Daten von Großrechner-Umgebungen über Speicherabbilder mit DTFJ zuzugreifen und nach Variablen und Werten zu untersuchen.

Sein Vorteil ist die Erweiterbarkeit und Anpassungsfähigkeit, sowie das Visualisieren von Situationen, die nicht mit Regelmäßigkeiten auftreten. In einem Debug-Modus können nur Fehler gefunden werden, die regelmäßig und an der gleichen Stelle auftreten.

Des Weiteren lässt sich so ein Programm um weitere Funktionalitäten wie eine Konfigurationsdatei, in der man Einstiegspunkte zur Suche, der Tiefe und die Objekte, nach denen gesucht werden soll, angibt, erweitern.

Ein wichtiger Punkt ist es, zum Zeitpunkt der Fehlfunktion die Daten und Zusammenhänge zu Systemeigenschaften darzustellen.

Dieses Projekt wird um eine Funktion erweitert, die die Identifikationsnummern der Objekte speichert und somit prüft, ob Objekte doppelt analysiert werden. Man möchte verhindern, in Loops durch Mehrfachreferenzierungen zu geraten. Eine ander Erweiterung stell die Ausgabe von Informationen zu Threads dar.

Nachdem das Ziel, das Extrahieren von Informationen von abgestürzten Programmen, erreicht wurde, wird überlegt, welche Funktionalität noch implementiert werden soll. Für die Zukunft stellt die komfortablen Bedienung durch eine grafische Oberfläche eine Aufgabenstellung dar.

Referenzen:

[1] Author: IBM, Person unbekannt
IBM Diagnostics Guide (pdf)
http://download.boulder.ibm.com/ibmdl/pub/software/dw/jdk/diagnosis/diag50.pdf

[2] Author: IBM, Person unbekannt
IBM Diagnostics Guide (pdf)
http://download.boulder.ibm.com/ibmdl/pub/software/dw/jdk/diagnosis/diag50.pdf
Seite 221/ 229
Zuletzt abgerufen am: 05. November 2008

[3] Author: IBM, Person unbekannt
IBM Diagnostics Guide (pdf)
http://download.boulder.ibm.com/ibmdl/pub/software/dw/jdk/diagnosis/diag50.pdf
Seite 229
Zuletzt abgerufen am: 05. November 2008

[4] Author: IBM, Person unbekannt
IBM Diagnostics Guide (pdf)
http://download.boulder.ibm.com/ibmdl/pub/software/dw/jdk/diagnosis/diag50.pdf
Seite 258
Zuletzt abgerufen am: 05. November 2008

[5] Author: IBM, Person unbekannt
IBM Diagnostics Guide (pdf)
http://download.boulder.ibm.com/ibmdl/pub/software/dw/jdk/diagnosis/diag50.pdf
Seite 373
Zuletzt abgerufen am: 05. November 2008