Wissenschaftliches Schreiben WS 2008/09

Autor: Gamif110
HfT-Stuttgart

Flight-Recorder

eine Applikation, die Speicherabbilder nach Objekten analysiert und diese anzeigt

14. Januar 2009: endgültige Abgabe

Inhaltsverzeichnis

1 Einleitung

Im ersten Kapitel geht es um die Einbettung des Themas in dessen Umfeld. Es wird erklärt, wie das Thema entstand.

1.1 Einleitung und Umfeld des Themas

Ein Flight-Recorder stellt ein Programm auf Konsolenbasis dar, welches Speicherabbilder von laufenden Java-Programmen analysiert und das Ergebnis am Bildschirm anzeigt oder in eine Textdatei schreibt.

Des Weiteren dient das Programm der Unterstützung der Fehlersuche. Es soll helfen, Speicherabbilder von Anwendungen nach unkontrollierten, nicht mit Regelmäßigkeit auftretenden Fehler zu untersuchen. Man versucht somit, an Informationen zur Behebung von nicht einfach erkennbaren Fehlerfällen, zum Beispiel Deadlocks oder unregelmäßige Stillstände (Hangs), zu kommen.
Nicht jeder Fehlerfall 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 von einem laufenden Programm analysiert und Variablen und Objekte nach ihren Inhalten prüft. Zuzüglich kann man sich die Konfiguration des Systems mit ausgeben lassen.

Speziell bei Fehlern, die nicht mit Regelmäßigkeit auftreten, ist es schwierig, mit einem regulären Analyseprogramm, einem Debugger, der den Quelltext durchläuft, dem Ursprung auf den Grund zu gehen.
Ein Flight-Recorder untersucht nicht den Quelltext aus einer Entwicklungsumgebung heraus, sondern die Daten, die während der Ausführung des Programms erzeugt und in den Variablen gespeichert wurden. Hier kann man genau auf die Daten des Fehlerfalls zurückgreifen, da man Speicherabbilder zu jedem Zeitpunkt der Ausführung erstellen und mit dem Flight-Recorder analysieren oder ausgeben kann.

Für Projekte, wie den Flight-Recorder, hat die Firma IBM für Ihre Version der Java Virtual Maschine (JVM) eine Klassenbibliothek, das Diagnostic Toolkit and Framework for Java, entwickelt, mit der man auf Speicherabbilder und deren Inhalte zugreifen kann.
Die Erstellung des Flight-Recorders stützt sich auf die Benutzung des Frameworks, welches nur in der IBM-Version der Java Virtual Machine (JVM) enthalten ist.

Es stellt sich die Frage, ob es möglich ist, mit dem Diagnostic Toolkit and Framework for Java (DTFJ) [1] ein Programm zu erstellen, das in der Lage ist, die gewünschten Daten aus Speicherabbildern laufender Programme zu extrahieren.

Spezielle Informatikbegriffe werden im Anschluss, im Teil Glossar, näher erläutert.

1.1. Begründung zur Entscheidung, mit dem Diagnostic Toolkit and Framework for Java zu arbeiten.

Anfangs war nicht klar, ob das Vorhaben, ein laufendes Programm aufzuzeichnen und auszuwerten, mit dem DTFJ möglich ist.

Ein einfaches Argument für den anfänglichen Fokus auf das DTFJ und den firmeninternen Werkzeugen ist, dass man nicht auf Fremdprodukte zurückgreifen sollte, so lange die eigenen weiterhelfen und gut genug sind, was es raus zu finden galt.

Unabhängig davon, hat man die Optionen sich Neuentwicklungen zu überlegen, vom Aufzeichnen des Programmablaufes bis zur Analyse, oder zunächst, mit dem, was man hat und die IBM-JVM schon bereitstellt, versuchen, an das Ziel zu gelangen. Dadurch, dass die IBM-JVM Speicherabbilder erstellen kann und es ein Werkzeug gibt, das Speicherabbilder allgemein zugänglich macht (JEXTRACT), lag es nahe, zunächst mit den vorhandenen Werkzeugen zu arbeiten. Damit waren die ersten Schritte erledigt.

Weitere Gründe zur Arbeit mit dem DTFJ beruhen auf die Klassenbeschreibungen. Hier wurden Klassen wie JavaObject, JavaField, JavaThread und JavaMonitor genannt und beschrieben. Aus den Informationen ging hervor, dass es möglich sein muss, Daten aus Speicherabbildern zu extrahieren.
Zusätzlich umfassen diese Klassen genau die Informationen, die man primär erarbeiten möchte.

Nach der Einbettung des Theamas, wird. Im Folgenden Kapitel erklärt, was Speicherabbilder sind, welche Typen es gibt und wie diese innerhalb einer Java-Umgebung erstellt werden können. Speicherabbilder stellen die Datengrundlage für den Flight-Recorder dar.

2 Speicherabbilder

In diesem Kapitel wird der Fokus zunächst auf Speicherabbilder gelegt, da diese die Datenquelle darstellen. Es werden Typen von Speicherabbildern erleutert, die sich mit der IBM-JVM erstellen lassen. Das wichtigste Speicherabbild ist das System dump, das dieser alle verfügbare Daten enthällt.

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

Von denen im Folgenden vorgestellten Speicherabbildern ist zur Analyse nur das System-Speicherabbild (System dump) interessant. Es enthält die meisten Informationen und ist somit am aussagekräftigsten. Somit stellt es die beste Chance dar, an die richtigen Daten zu gelangen [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 verfügbaren Java-Objekte, die sich im Java Heap befanden. Diese Art von Speicherabbild erzeugt eine binär komprimierte Datei.

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 widerspiegeln. Auf einer z-Architektur kann sich ein Programm über mehrere Adressräume erstrecken. Diese Rechner besitzen sehr große Speicher. Es kann somit vorkommen, daß ein Speicherabbild auf einer z-Maschine mehrere Gigabyte groß ist. Auf Windows-Architekturen sind die Adressräume pro Prozess auf 1 begrenzt und der Arbeitsspeicher kleiner.

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 (JVM) 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].

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

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

Tool
Mit Hilfe dieser Option lässt sich eine externe Anwendung/ Prozess starten, wenn ein bestimmtes Ereignis auftritt. Man kann damit bestimmte Nachrichten auf dem Bildschirm anzeigen lassen oder eine Konsole starten.

Console
Mit der Option Console lässt sich ein sehr einfaches und kleines Abbild eines Java-Programmes generieren. Es enthält nur den Namen und den Status eines jeden Thread, Semaphore zu den Objekten und den Eigentümern derer.

Snap:
Snap traces werden durch die Xdump-Option kontrolliert. Sie beinhalten die Daten der gesetzten tracepoints im Trace-Puffer.

Heap:
Die Option Heap veranlasst, dass ein Heap dump erstellt wird. Dieser enthält Daten über alle Java Objekte, die sich zur Zeit der Generierung im Heap befanden und nicht von der Garbage Collection aus dem Speicher gelöscht wurden.

Java:
Java dumps sind intern generierte und formatierte Berichte, die ohne weitere Werkzeuge betrachtet und analysiert werden können. Diese Option bietet Informationen über gestartete Threads, geladene Klassen und Statistiken über den Heap.

System.
Mit dieser Option wird der komplette Addressraum eines Programmes abgebildet. Ersellt wir es im Binärformat. Somit sind, mit dieser Option erstellte Abbilder, sehr groß. Bei Servern oder Großrechnern können diese bis zu mehreren Gigabyte groß sein.

Die nächste Option in der Anweisung stellt die Events (Ereignisse) dar, für die 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

Dump durch Java-Anweisung

Im Package com.ibm.jvm.Dump der IBM-Java-Umgebung sind folgende Funktionen/ Methoden untergebracht, die einen Dump auslösen:

- com.ibm.jvm.Dump.SystemDump()
  Mit dieser Methode kann ein System-Speicherabbild aus einem laufenden Java-Programm angestoßen werden
- com.ibm.jvm.Dump.HeapDump()
  Mit dieser Methode wird die Erstellung eines Heap dumps angestoßen
- com.ibm.jvm.Dump.JavaDump()
  Mit dieser Methode wird die Erstellung eines Java dumps angestoßen

Diese Funktionen lassen sich aus einem Java-Programm heraus starten.
Der Status der im Programm befindlichen Daten ist der zur Ausführung der Dump-Anweisung. Für die Entwicklung des Flight-Recorders ist die Anweisung com.jbm.jvm.Dump.SystemDump() die wichtigste und wurde ausschließlich 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.

2.4. Vorbereitung mit JEXTRACT

Um System-Speicherabbilder unabhängig der Maschine, auf der sie erstellt werden, 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 (Diagnostic Toolkit and Framework for Java) gelesen werden. Unterschiedliche Wortgröße, Byte-Reihenfolge und Arten von Datenstrukturen erschweren es, ohne Vorbereitungen Informationen verschiedener Systeme bereitzustellen. Zur Bereitstellung der Informationen der Dumps, müssen sie zunächst 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 beendet sind, kann man durch die DTFJ-API auf den Speicher des zu untersuchenden Programms zugreifen [4].

Anweisung für Ausführung von JEXTRACT:
Jextract <System dump> <optional neuer Dateiname>

Nachdem ein System dump mit JEXTRACT vorbereitet wurde, kann er einem mit dem DTFJ-API erstellten Programm übergeben werden und dieses das Speicherabbild analysiert werden.

Im letzen Kapitel wurde beschrieben, wie ein System-Speicherabbild erstellt wird. Nachdem die Grundlage des Flight-Recorders, ein Abbild eines laufenden Programmes, ersteltl worden ist, kann man im nächsten Kapitel das Programm erleutern, das dieses Abbild analysiert.

3 Erstellung des Flight-Recorders

Im 3. Kapitel geht es um die Programmierung des Flight-Recorders, unter zur Hilfenahme des von IBM entwickeltem Klassenbibliothek DTFJ. Es wird der Zusammenhang der verschiedenen Klassen, sowie die anschließende Darstellung der Daten erläutert.

3.1. Vorgehensweise mit DTFJ

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

Die Technik gibt die Möglichkeit, über verschachtelte while-Schleifen und 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. Um herauszufinden welche Variablen ein Objekt hat, geht man den Weg über dessen Definition, über seine Klasse.

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 darunter liegender Ebene bedient und den Zugriff darauf ermöglicht.

3.2. Allgemeine Hinweise zur Programmierung

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

Es ist nicht möglich, mit einer einfachen Methode direkt auf die Werte angelegter Objekte und Variablen zuzugreifen und sie anzeigen zu lassen. Um auf Objekte zuzugreifen, muß eine verschachtelte Struktur aus while-Schleifen aufgebaute werden und es ist notwendig, verschiedene höher stehende Ebenen zu durchlaufen. Aus Programmierersicht stellt eine Ebene eine Klasse dar. Die Klassen sind dadurch miteinander verbunden, dass sie Instanzen der tiefer liegenden Klassen enthalten. Um auf die Daten von Objekte und Variablen zuzugreifen, muss man mehrere dieser Instanzen/ Ebenen durchlaufen. Da es möglich ist, dass manche Instanzen mehrfach vorkommen, wird ein Iterator zurückgegeben, der auf alle Instanzen der Klasse zeigt.

Ein kleines Beispiel:
Da ein Programm mehr als ein Adressraum enthalten kann, muß man auf der Suche nach den gewünschten Variablen, jeden Adressraum durchlaufen. Jeder Adressraum enthält mehrere Java Laufzeitumgebungen, die man wiederum alle aufrufen muß. Innerhalb jeder Laufzeitumgebung existieren mehrere JavaHeaps. Diese müssen auch alle aufgerufen werden und zu einer Klasse zu gelangen, die die gewünschten Daten enthält. Es handelt sich um die Klasse JavaObjekt. Die Objekte dieser Klasse enthalten die Daten des Programms, das analysiert werden soll bzw. von dem ein Speicherabbild erstellt wurde.

In jeder Klasse gibt es einen Iterator, der die Objekte der nächten Klasse/ Ebene enthalten. Innerhalb jeder Ebene wird einen while-Schleife erstellt, die alle Instanzen der unteren Ebene aufruft. Dies wiederholt sich, bis man sich auf der Ebene der Objekte befindet.

In den folgenden Abschnitten wird näher erleutert, welche Daten auf den einzelnen Ebenen darstellbar sind, wie sie miteinander verbunden sind und wie man auf Objekte zugreift.

3.3. Bereitstellung des Speicherabbildes für das DTFJ

Der erste Schritt ist, dem DTFJ den Zugriff auf das komprimierte Verzeichnis mit dem Speicherabbild und der dazugehörigen XML-Datei zu ermöglichen. Man übergibt den Pfad zur, mit JEXTRACT erstellten Datei, einem File-Objekt. Hierfür lassen sich die Methoden und Konstruktoren der File-Klasse zur Erstellung von File-Objekten benutzen.

Im nächsten Schritt wird mittels Reflection ein ImageFactory-Objekt kreiert. Mittels ImageFactory-Object wird das File-Objekt an ein statisch angelegtes Image-Objekt übergeben, welches den Einstieg in die Analyse des Speichers darstellt.

Die einzelnen Schritte sind folgende:
Es wird ein Objekt namens factoryClass der Klasse Class angelegt.
Instanziert wird es mit der Methode Class.forName(„com.ibm.dtfj.image.j9.ImageFactory“)
Im nächsten Schritt wird ein Objekt namens factory der Klasse ImageFactory angelegt und mit der Anweisung (ImageFactory) factoryClass.newInstance() des zuvor erstellten Objektes instanziert.
Der dritte Schritt ist es, das Image zu instanzieren.
Dies passiert mit der Anweisung image = factory.getImage(File-Objekt)

Diese drei Schritte sollten von einem try/catch-Block eingeschlossen werden, da es beim Erstellen dieser zu bestimmten Fehlern kommen kann. In der Catch-Anweisung sollten die Ausnahembehandlungen zur ClassNotFoundException, IllegalAccessException, InstantiationException und IOException abgefangen werden.

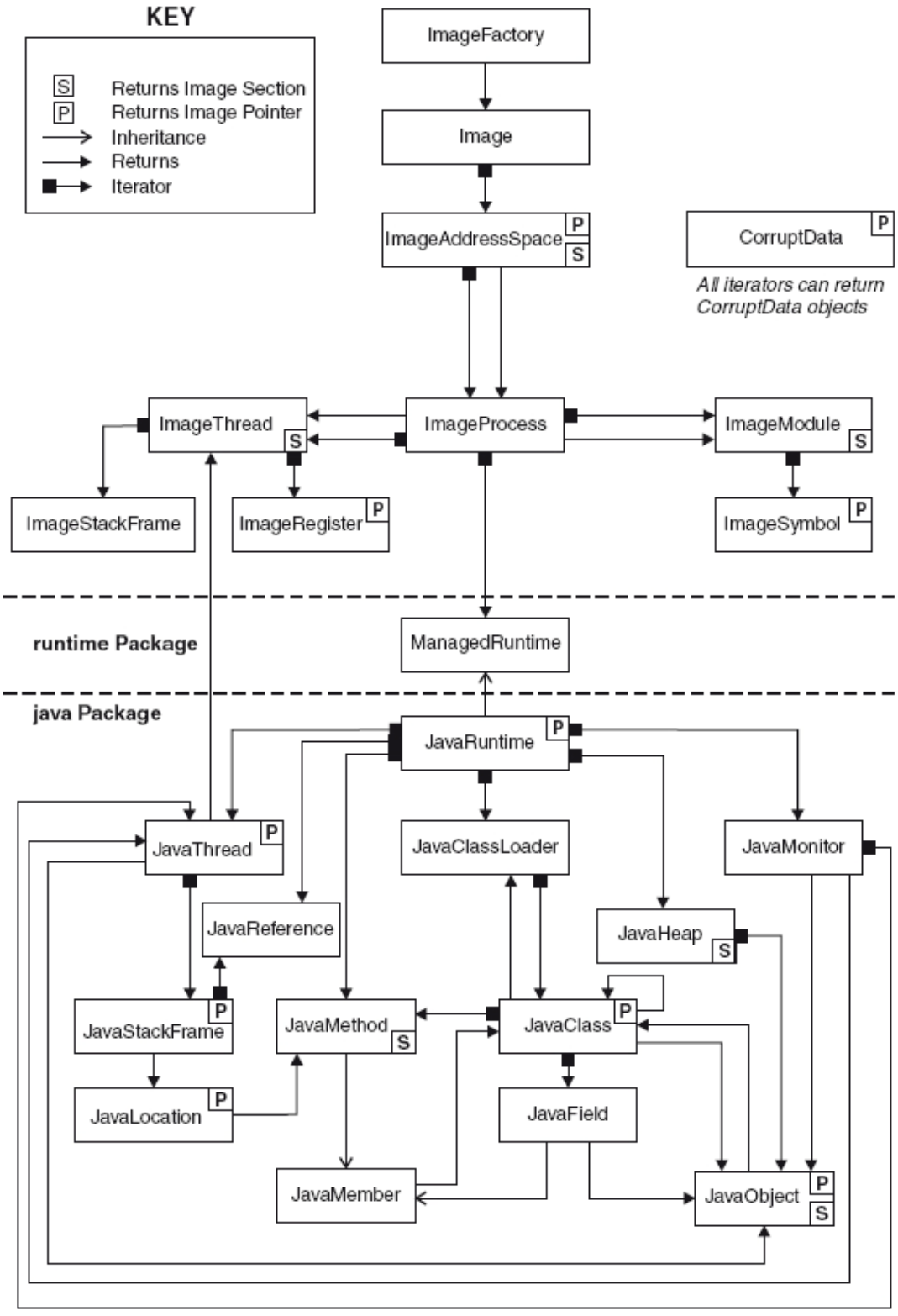
Code-Beispiel [CB]:

```
File f = new File( args[0] );
        try
        {
           Class factoryClass = Class
                .forName( "com.ibm.dtfj.image.j9.ImageFactory" );
           ImageFactory factory = (ImageFactory) factoryClass.newInstance( );
           image = factory.getImage( f );
        }
        catch ( Exception ex )
        { /*
            * Should use the error handling as shown in DTFJEX1.
            */
           System.err.println( "Error in DTFJEX2" );
           ex.printStackTrace( System.err );
        }
```

3.4. Aufbau der Klassen

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

Die folgende Grafik zeigt die Zusammenhängigkeit der einzelnen Klassen und welche Objekte erstellt werden müssen, um an bestimmte/ darunter liegende Informationen zu gelangen [KG].



3.5. Auslesen von Variablen

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 enthält. Im Schleifenkörper wird durch die Methode Iterator.next() ein Objekt des Adressraumes angelegt. Dieses Objekt lässt sich nach Informationen abfragen. Innerhalb der Klasse ImageAdressSpace kann man noch nicht auf angelegte Objekte und Variablen zugreifen. Man muss sich den Iterator zu ImageProcess zurückgeben lassen und diese Klasse als nächstes ansteuern, um zu Variablen zu gelangen.

Zugriff auf Prozesse

Jedem Adressraum sind Prozesse zugeteilt. Diese stellen die nächste zu überbrückende Ebene dar, um Objekte abzufragen. Um Objekte abfragen zu können, werden die Prozesse abgefragt und durchlaufen. Dazu stellt das Framework dieselben Instrumente zur Verfügung, wie auf der Ebene der Adressräume. Innerhalb der while-Schleife der Adressräume legt man mit Hilfe der Methode getProcesses() einen Iterator an. Mit diesem Iterator und einer weiteren while-Schleife erstellt man mit jedem Iteratordurchgang ein Objekt der Klasse ImageProcess.
Die Objekte der Klasse ImageProcess stellen verschiedene Methoden zur Verfügung. Keine dieser stellt den Zugriff auf Variablen dar. Hier muss eine Verbindung zu der Klasse JavaRuntime hergestellt werden.

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 weiterzählen lässt. Innerhalb des Körpers wird ein Objekt der Klasse JavaRuntime angelegt. Auch diese Ebene stellt noch kein Zugriff auf Objekte oder Variablen zur Verfügung. Auch hier muß eine Verbindung zu einer weiteren Klasse aufgebaut werden. Die letzte Zwischenstation, um auf Objekte zuzugreifen, ist die Klasse JavaHeap.

Zugriff auf Heaps

Die Ebene der JavaRuntime ermöglicht den Zugriff auf Objekte der Klasse JavaHeap. Mit der Methode JavaRuntime.getHeaps() bekommt man durch einen Iterator den Zugriff auf die einzelnen Heaps. Die Klasse JavaHeap stellt eine direkte Verbindung zu der Klasse JavaObject, die den Zugriff auf Objekte des zu analysierenden Programms dar.

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 wiederholt. Innerhalb einer While-Schleife wird ein Objekt der Klasse JavaObjekt angelegt. Diese spiegeln die Objekte des als Speicherabbild übergebenen Programms wider. 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. Nun befindet man sich auf der Ebene der Objekte, deren Variablen angezeigt werden können. Es ist nicht alleine mit der Klasse JavaObjekt möglich, Variablen anzuzeigen. Man benötigt außer der Klasse JavaObjekt, die Klasse der Variablen, JavaField, und noch eine weitere Klasse, die das Objekt beschreibt. Das übernimmt die Klasse JavaClass. Diese Klasse zeigt die Struktur des Objektes und somit auch welche Variablen Objekte enthalten

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 eines Objekts
- JavaObject.getSize() liefert die Größe eines Objektes

.

Zugriff auf JavaClass

Der Zugriff auf Objekte der Klasse JavaClass ist wichtig, um Felder und Methoden auslesen zu können. Diese Klasse beschreibt die Struktur der Objekte und kennt den Namen der Variablen, die ein Objekt enthält. Mit der Methode JavaObject.getJavaClass() erhält man eine Instanz der Klasse, die das Objekt beschreibt und kann diese nach den Namen der deklarierten Felder des Objektes fragen.

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

Felder eines Objektes abfragen

Der Zugriff auf Felder und Methoden ist durch die Klassen JavaObject, JavaClass und JavaField realisiert. Die Klasse JavaClass besitzt die Methode JavaClass.getJavaFields(), die einen Iterator zu den Feldern, die innerhalb eines Objektes angelegt wurden, zurückgibt. Mit diesem Iterator kann in einer weiteren Schleife jedes Feld zurückgegeben werden. Die zurückgegebenen Felder sind Objekte vom Typ JavaField und enthalten die Daten, die am Bildschirm ausgegeben werden sollen.

Werte von Variablen zurückgeben lassen

Um an die Daten von Feldern heranzukommen, muß man die Klassen JavaObject, JavaClass und JavaField gemeinsam bedienen. Innerhalb der Schleife die Felder eines Objektes durchläuft, Klasse JavaClass, fragt man jedes Feld nach seinem Typ. Dies geschieht mit der Methode JavaField.getSignature(). Die Signatur gibt den Typ einer Variablen an. Je nach Signatur gibt es eine andere Methode um den Wert der Variablen zurückzugeben. Um die korrekte Methode zu einer Signatur aufzurufen, eignet sich der Aufbau einer If/Else-Struktur, in der man ein Feld zunächst nach seiner Signatur abfragt und darauf durch if/else-Abfragen die richtige Methode startet.

Folgende Methoden sind für Zugriff auf elementare Datentypen zuständig:

- Boolean: JavaField.getBoolean(JavaObject)
- Byte: JavaField.getByte(JavaObject)
- Short: JavaField.getShort(JavaObject)
- Integer: JavaField.getInt(JavaObject)
- Long: JavaField.getLong(JavaObject)
- Float: JavaField.getFloat(JavaObject)
- Double: JavaField.getDouble(JavaObject)
- Char: JavaField.getChar(JavaObject)
- String: JavaField.getString(JavaObject)

Der Übergabeparameter JavaObject ist das Objekt, von dem zuvor die Klasse JavaClass aufgerufen wurde. Um einen Wert einer Variablen auszulesen, muss immer ein JavaField-Objekt über die Klasse JavaClass erstellt werden und die Funktion des richtigen Typs, mit Übergabe des aktuellen Objektes, aufgerufen werden.

Weitere Methoden der Klasse JavaField:

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

Darstellung der Variablen

Zur Darstellung der Daten hat man sich auf eine sehr einfache Möglichkeit beschränkt. Sobald man auf Daten in Variablen stößt, wird eine Ausgabe-Methode aufgerufen, die wiederum eine einfache Stringausgabe-Methode startet und den Namen der Variable und dessen Wert ausgibt. Da das Programm als Konsolenanwendung ausgeführt wird, werden die gefundenen Daten mit einer einfachen Ausgabe, System.out.println(String), auf der Konsole dargestellt.
Als Option kann man die Bildschirmausgabe, Beispiel Windows und Linux, in eine Textdatei umlenken. Hierfür dient der Operator ‚>':

Beispiel: „Programmaufruf" > „Name Textdatei"

4 Zusammenfassung, Fazit und Ausblick

Zunächst muss ein System-Dump nach einem Programmfehler erstellt werden. Dieser muß 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 Möglichkeit darstellt, auf Daten von laufenden Programmen über Speicherabbilder zuzugreifen. und deren Variablen darzustellen..

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 ein Analyse-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.

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 Endlosschleifen durch Mehrfachreferenzierungen zu geraten. Eine andere Erweiterung stellt 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 ist die Erstellung einer grafischen Bedienoberfläche eine Aufgabenstellung dar.

5 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

[CB] Author: IBM, Person unbekannt
IBM Diagnostics Guide (pdf)
http://download.boulder.ibm.com/ibmdl/pub/software/dw/jdk/diagnosis/diag50.pdf
Seite 374

[KG] Author: IBM, Person unbekannt
IBM Diagnostics Guide (pdf)
http://download.boulder.ibm.com/ibmdl/pub/software/dw/jdk/diagnosis/diag50.pdf
Seite 376

6 Glossar

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

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

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

Java:
Java ist eine objektorientierte Programmiersprache, die in den 90er Jahre entwickelt wurde und zur Firma Sun gehört. Mit dieser Sprache kann plattformunabhängig programmiert werden, so dass die darin entwickelten Programme auf allen Betriebssysteme laufen.

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. Dadurch müssen Programme für verschiedene Plattformen/ Betriebssysteme nicht neu kompiliert werden.

z/OS:
Großrechner-Betriebssystem der Firma IBM

Trace:
Traces dienen der Spurenverfolgung der Fehler. Es werden Anhaltspunkte zum Verlauf des Programms ausgegeben.

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

Framework:
Ein Framework stellt eine Funktionalität dar, die für das vorgesehene Programm nicht erst erstellt/ programmiert werden muß, sondern schon zur Verfügung steht.

Prozess:
Ein Prozess innerhalb eines Systems ist eine Einheit der Tätigkeit des Systems. Dieser umfasst Programmcode, Kontext und befindet sich zu jedem Zeitpunkt in einem genau definierten Zustand.
Aus Betriebssystemsicht ist ein Prozess ein Programm, das sich in Ausführung befindet und dem Betriebsmittel zugeteilt sind.

Thread:
Ein Thread ist ein Teil eines Prozesses, ein Unterprozess. Er kann eigenständig innerhalb eines Prozesses laufen. Unterprozesse teilen sich den Adressraum, der dem Prozess zugeteilt wurde.

Semaphore/ Monitore:
Ein Semaphor ist eine Datenstruktur, bestehend aus Warteschlange und Zähler, die bei synchroner Ausführung mehrerer Prozesse/ Threads Störfälle vermeiden soll. Es können Wechselwirkungen auftreten, wenn z. B. mehrere Prozesse gleichzeitig auf dieselben Dateien zugreifen möchten. Dieser Zugriff muß sequenziell geregelt werden. Diese Aufgabe übernehmen Semaphore mit ihrer Warteschlange und dem Zähler.

Deadlock/ Verklemmungen:
Ein Deadlock stellt eine Situation dar, wenn ein oder mehrere Prozesse auf Betriebsmittel warten, die von anderen beteiligten Prozesse blockiert werden und sie deshalb nicht mehr weiter arbeiten können.

Heap:
Als Heap bezeichnet man einen dynamischen Teil eines Speichern, in der Regel Teil des Arbeitsspeichers. Man kann in beliebiger Reihenfolge Speicher anfordern und wieder freigeben.

Stderr-Kanal:
Die Ein- und Ausgabe auf Konsolenbasis arbeitet mit Ströme und Kanäle. Es gibt drei Kanäle: Eingabe, Ausgabe und Standard-Error (Stderr). Für jeden Kanal kann das medium festgelegt werden, auf das es reagiert. Standardmäßig ist festgelegt, dass Eingabe Tastatur, Ausgabe und Stderr Bildschirm sind.

XML-Datei:
Eine XML-Datei stellt eine Textdatei dar, die Text enthällt, der mit einer HTML-ähnlichen Auszeichnungsprache formatiert ist.