2024_jan_01

Razhroščevanje in razhroščevalniki

A razhroščevalnik/debugger oz. razhroščevalno orodje/debugging tool je računalniški program, prvenstveno namenjen testiranju in razhroščevanju drugih programov (t.i. ciljnih/"target" programov).
V osnovi debugger izvaja ciljni program v kontroliranem/znanem(beri:predvidljivem) okolju, ki programerju dopušča sledenje izvajanju programa, uvid v spremembe v programu med izvajanjem, in vire v računalniku, ki lahko vplivajo na izvajanje oz. na katere izvajanje vpliva neposredno ali posredno.

Tipičen debugger tako omogoča:

- zagon in zaustavitev ciljnega programa na določeni točki v programu - omejevanje področja sledenja
- prikaz stanja pomnilnika, registrov CPE, trajnega pom. medija(npr. diska, .. - sledenje
- spremembo vsebine pomnilnika, dela pomnilnika ali registra - vnos testnih podatkov
- izvajanje po korakih (step by step)

Za doseganje enega ali vseh ciljev:

- : dokazovanje ustreznosti/pravilnosti izvajanja
- : preskus trpežnosti z vnosom neustreznih podatkov
- : iskanje napak v izvajanju in odpravljanje napak
- :: spoznavanje delovanja programa :=> crakcing
- ::'meritev' porabe prostora, časovnosti postopkov (določanje učinkovitosti)

Boljši omogočajo tudi:
- snemanje izvajanja in ponovno predvajanje
- obrnjeno razhroščevanje : koraki nazaj (IntelliTrace/sense)
- time travell debugging : izvedeš, spremeniš pogoje, greš nazaj, še enkrat izvedeš, ...

določeni procesorji omogočajo strojno debugiranje (npr. ARM: JTag..)

Java
JDK privzeto vsebuje debugger, imenuje se jdb in se nahaja v bin direktoriju distribucije. V izhodišču je to interaktivno orodje. Več o rabi lahko najdete: Oracle, The jdb Command,2023, https://docs.oracle.com/en/java/javase/21/docs/specs/man/jdb.html .
V večino razvojnih okolij (IDE) je razhroščevalnik vgrajen v eni od oblik, več ali manj imajo vsi grafični uporabniški vmesnik(Eclipse, NetBeans,IntelliJ, …). Pomoč za VS Code: https://code.visualstudio.com/docs/java/java-debugging .

1. Na dveh primerih:
   a) Test_01_lin
   b) Test_02_met,

   bomo uporabili interaktiven debugger JDB. Za lažji začetek je tule podane del sekvence izvedbe. Ker je mogoče debugger uporabiti zgolj na prevedenih programih, pazite, da boste uporabili prevajalnik in debugger iz iste distribucije Java. Pot do distibucije Java poiščite sami, spodaj je podan zgolj primer.

2024_jan_01

Razhroščevanje in razhroščevalniki

1.a)

C:\Users\dijak\JDB_test>\BlueJ500\jdk-21\bin\javac Test_01_lin.java

C:\Users\dijak\JDB_test>\BlueJ500\jdk-21\bin\jdb Test_01_lin.java
Initializing jdb ...
> **stop in Test_01_lin.main**
Deferring breakpoint Test_01_lin.main.
It will be set after the class is loaded.
> **run**
run Test_01_lin
Set uncaught java.lang.Throwable
Set deferred uncaught java.lang.Throwable
>
VM Started: Set deferred breakpoint Test_01_lin.main

Breakpoint hit: "thread=main", Test_01_lin.main(), line=7 bci=0
7 a = 10;

main[1] **list**
3
4 public static void main(String[] args){
5
6 int a,b;
7 => a = 10;
8 a = a + 1;
9 b = 2;
10
11 for (int i=0;i<5;) {
12 a = a + 1;
main[1] **print a**
com.sun.tools.example.debug.expr.ParseException: Name unknown: a
a = null
main[1] **step**
>
Step completed: "thread=main", Test_01_lin.main(), line=8 bci=3
8 a = a + 1;
main[1] **print a**
a = 10
main[1] **stop at Test_01_lin:11**
Set breakpoint Test_01_lin:11
main[1] **cont**
>
Breakpoint hit: "thread=main", Test_01_lin.main(), line=11 bci=9
11 for (int i=0;i<5;) {

main[1]

Razhroščevanje in razhroščevalniki

Izvedite zgornji primer. Sled(trace) izvedite korak za korakom tudi preko celotne zanka. V poročilo tega dela naloge dodajte zaslonsko sliko rezultata ukaza list pri i==3.

Debagirati je možno tudi zgolj .class datoteko brez izvorne kode, je pa res, da v postopku potem dobivamo manj informacij.

1.b)
Izvedite 'debugging' programa Test_02_met. Postopek naj gre kot: zaženite program in ugotovite, ali izpis(rezultat) ustreza specifikaciji programa(metod). Če ustreza, 'debagiranje' ni potrebno. Če ne, ugotovite, kako bi najučinkoviteje in najhitreje odkrili napako, 'dokažite jo z dumpom v debagerju', nato napako odpravite. V poročilo tega dela gresta list mesta vsaj ene izmed napak, ter dump spremenljivke z napačno vrednostjo.

1.c) Odgovorite na vprašanja

1.1 Kaj se zgodi, če po vstopu v debugger namesto **stop in Test_01_lin.main** uporabite **run ?**
1.2 Kaj povzroči ukaz **list** in ali ga lahko uporabite predenj zaženete **run** ?
1.3 Kakšna je razlika med **print a** in **dump a** ?
1.4 Kaj je 'prekinitvena točka' (breakpoint), kdaj bi lahko bila koristna in s katerim ukazov jo nastavimo?
1.5 Kako nastavite breakpoint na specifični vrstici, kako na prvi ukaz v izbrani metodi?
1.6 Kaj dosežete z ukazom step / next ?
1.7 Kako dosežete nadaljevanje izvajanje po predhodni zaustavitvi na breakpointu in bi želeli izvajanje nadaljevati do naslednjega breakpointa ?
1.8 Kako izstopite in metode in se vrnete na izvajanje v klicoči program?
1.9 Kako dostopite do pomoči znotraj **jdb** ?
1.10 Ali je z debuggerjem v javanskih program možno poiskati in odpraviti tudi sintaktično napako? (kako? oziroma zakaj?)

2. V prilogi 3 je dan še en primer. Določite mesta napak; v poročilo dodajte zaslonske slike reultatov ukazov list in dump(print) na mestih napak, ki jasno, kažejo na napako.

3. Ponovite izvedbe nalog 1 in 2 še z v IDE (BlueJ) vgrajenim 'debugerjem' brez ponovnega odgovarjanja na vprašanja iz 1c; breakpoint tipično postavljate s klikom na trak levo pred vsebino vrstice. Pomni, da lahko točko prekinitve postavite zgolj na izvedbeno efektivno vrstico (kodo, ne npr. na ime funkcije, programa, …)

2024_jan_01

Razhroščevanje in razhroščevalniki

Priloga 1

```
public class Test_01_lin{

    public static void main(String[] args){

        int a,b;
        a = 10;
        a = a + 1;
        b = 2;

        for (int i=0;i<5;) {
            a = a + 1;
            i++;
            System.out.println("a: "+a+" i:"+i);
        }

    }
}
```

Priloga 2

```
public class Test_02_met{

    public static void main(String[] args){
        int a,b;
        a = 10;
        b = 20;
        a = a + vsota(a,b); //40
        a = a + petKratnik(a);  // 6*a - 6*40 = 240
        b = petKratnik(b);  //5*20 ?
        System.out.println(a+" "+b);
    }

    /**
     *   vrne vsoto dveh argumentov
     */
    static int vsota(int a, int b){
       int vst = 0;
       vst = vst + a + b;
       return vst;
    }

    /*
     *   brez rabe množenja
     */
    static int petKratnik(int a){
        a=0;
        for(int i=1;i<4;i++)
           a +=a;
        return a;
    }
}
```

2024_jan_01

Razhroščevanje in razhroščevalniki

Priloga 3

Java, koda je bila ustvarjena z UI (ChatGPT4). Pozorno preglejte in uporabite:

> *Za vas sem tudi ustvaril testni javanski program, ki vsebuje nekaj napak, ki jih lahko poskusite popraviti z JDB. Program je namenjen izračunu kvadratne enačbe, ki ima obliko ax^2 + bx + c = 0.*
> *Program sprejme tri parametre a, b in c in izpiše rešitve enačbe. Program lahko zapišete v datoteko z imenom KvadratnaEnacba.java in ga prevedete z ukazom javac KvadratnaEnacba.java. Nato ga lahko zaženete z ukazom jdb KvadratnaEnacba in sledite navodilom na zaslonu.*

Mimogrede debuger zaganjate enako kot interpreter; torej *jdb KvadratnaEnacba 12.3 17.0 3.3* in pa: močno zaupam nezmotljivi UI, tako da spodnjega nisem niti preveril.

```
public class KvadratnaEnacba {

    public static void main(String[] args) {
        // Preberemo parametre a, b in c iz ukazne vrstice
        double a = Double.parseDouble(args[0]);
        double b = Double.parseDouble(args[1]);
        double c = Double.parseDouble(args[2]);

        // Izračunamo diskriminanto
        double d = b * b - 4 * a * c;

        // Preverimo, ali ima enačba realne rešitve
        if (d >= 0) {
            // Izračunamo rešitvi
            double x1 = (-b + Math.sqrt(d)) / (2 * a);
            double x2 = (-b - Math.sqrt(d)) / (2 * a);

            // Izpišemo rešitvi
            System.out.println("Rešitvi enačbe " + a + "x^2 + " + b + "x + " + c + " = 0 sta:");
            System.out.println("x1 = " + x1);
            System.out.println("x2 = " + x2);
        } else {
            // Enačba nima realnih rešitev
            System.out.println("Enačba " +a+ "x^2 + " + b + "x + " + c + " = 0 nima realnih rešitev.");
        }
    }
}
```

2024_jan_01

Razhroščevanje in razhroščevalniki

Prostor za zapiske/rešitve