Threads i Java

Denne artikel giver en introduktion til threads i Java.

Den beskriver hvad tråde er og forklarer hvordan de bruges i Java

Den forudsætter kendskab til Java men ikke til threads.

Skrevet den **16. Feb 2010** af **arne_v** I kategorien **Programmering / Java** | ★★★☆☆

Historie:
V1.0 - 12/01/2004 - original
V1.1 - 31/01/2004 - forbedret formatering + uddybning af de 2 måder
V1.2 - 07/04/2004 - tilføj note om stop af tråd
V1.3 - 05/08/2005 - tilføj Java 1.5 thread pool eksempel
V1.4 - 16/02/2010 - smårettelser

Hvad er threads

Threads (eller tråde på dansk) er er et begreb som dækker over flere logiske eksekverings flow inden for samme program.

Groft sagt kan programmet "lave flere ting samtidigt".

Begrebet er ikke Java specifikt. Både Windows Threads og POSIX Threads er ældre. Men Java er designet fra starten af til at understøtte threads.

En applikation med kun en tråd kaldes for en single threaded applikation, mens en med flere tråde kaldes for en multi threaded applikation.

Der er 2 low level måder at starte tråde på i Java. Jeg vil kort vise begge.

Simple eksempler

Eksempel med extends Thread:

```
// demo klasse
public class T1 {
    public static void main(String[] args) throws Exception {
        // lav tråd objekt
        FirstWay t = new FirstWay();
        // start tråd
        t.start();
        // vent på tråd afslutter
        t.join();
```

```
    }
}

// klasse der repræsenterer tråd
// skal extende Thread
class FirstWay extends Thread {
    // metode der kaldes når tråden starter
    // skal hedde run
    public void run() {
        System.out.println("Tråd kører");
    }
}
```

Eksempel med implements Runnable:

```
// demo klasse
public class T2 {
    public static void main(String[] args) throws Exception {
        // lav tråd objekt
        Thread t = new Thread(new SecondWay());
        // start tråd
        t.start();
        // vent på tråd afslutter
        t.join();
    }
}

// klasse der repræsenterer tråd
// skal implementere Runnable
class SecondWay implements Runnable {
    // metode der kaldes når tråden starter
    // skal hedde run
    public void run() {
        System.out.println("Tråd kører");
    }
}
```

Jeg plejer at bruge den første, fordi jeg synes at det er mere indlysende hvad der sker. Specielt hvis man bruger this som argument til new Thread i den anden synes jeg at koden er forvirrende.

Den anden har imidlertid en fordel nemlig at man kan bruge den selvom klassen i forvejen extender en klasse.

Eksempel med mange tråde

Hvis man skal starte flere tråde end en, så er følgende teknik tit brugbar:

```
public class T3 {
    private final static int N = 10;
    public static void main(String[] args) throws Exception {
        MultiThread[] t = new MultiThread[N];
        for(int i = 0; i < N; i++) {
            t[i] = new MultiThread(new Info(i));
        }
        for(int i = 0; i < N; i++) {
            t[i].start();
        }
        for(int i = 0; i < N; i++) {
            t[i].join();
        }
    }
}

class MultiThread extends Thread {
    private Info inf;
    public MultiThread(Info inf) {
        this.inf = inf;
    }
    public void run() {
        System.out.println(inf.getV());
    }
}

class Info {
    private int v;
    public Info() {
        v = 0;
    }
    public Info(int v) {
        this.v = v;
    }
    public int getV() {
        return v;
    }
    public void setV(int v) {
        this.v = v;
    }
}
```

Ting man skal være opmærksom på

Der er nogle ting som man skal tage højde for i en multithreaded applikation.

Man kan komme i problemer hvis to tråde arbejder på de samme data samtidigt.
En tråd kan nemlig blive swappet ud midt i noget og en anden tråd kan starte.
Og hvis den anden tråd modificerer data som den første tråd bruger, så kan det
give sære fejl.

Det er endnu mere risikabelt på fler CPU maskiner, hvor flere tråde kan køre parallelt.

Det kan være en static klasse variabel eller en normal instans variabel i et objekt som begge tråde bruger. Lokale variable indeni metoder er der aldrig problemer med.

Java har imidlertid nogle indbyggede funktioner til at styre det.

Hvis objektet der skal tilgåes fra flere tråder hedder o kan putte de kritiske kode omårder inden i:

```
syncronized(o) {
    ...
}
```

så vil den pågældende kode kun blive udført af en tråd af gangen for det objekt.

```
synlighed synchronized returtype metodenavn(...) {
    ...
}
```

har samme betydning som:

```
synlighed returtype metodenavn(...) {
    synchronized(this) {
        ...
    }
}
```

og er altså bare en nemmere måde at skrive det på.

synchronized på en static metode synkroniserer tilsvarende på klassen fremfor instansen.

Der findes også mere avancerede mekanismer i form af wait og notify/notifyAll metoderne. Mit råd er at undgå dem hvis muligt, da de godt kan drille en del.

Bemærk at join metoden venter på at tråden afslutter af sig selv. Mens stop metoden er deprecated. Hvis man vil afbryde en tråd i utide, så skal man bruge interrupt metoden.

Nyt i Java 1.5

Der er en lang række spændende nyheder i Java 1.5 om threading

i java.util.concurrent pakken.

Bl.a. er der indbygget support for thread pool.

Her er en lille appetit vækker:

```
import java.util.Date;
import java.util.concurrent.Callable;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;
import java.util.concurrent.TimeUnit;

public class ThreadPool {
    public static void main(String[] args) throws InterruptedException,
ExecutionException {
        ExecutorService es = Executors.newFixedThreadPool(2);
        int nowork = 10;
        Future<Integer>[] workres = new Future[nowork];
        for(int i = 0; i < nowork; i++) {
            workres[i] = es.submit(new SomeWork(i));
        }
        for(int i = 0; i < nowork; i++) {
            System.out.println(workres[i].get());
        }
        es.shutdown();
    }
}

class SomeWork implements Callable<Integer> {
    private int v;
    public SomeWork(int v) {
        this.v = v;
    }
    public Integer call() throws Exception {
        TimeUnit.SECONDS.sleep(5);
        System.out.println(v + " done at " + (new Date()));
        return v;
    }
}
```

Det kan anbefales at bruge java.util.concurrent klasser fremfor egne tråd klasser hvis muligt.

Der er også et antal smarte data strukturer bl.a. køer til kommunikation mellem tråde.

Kommentar af simonvalter d. 31. Jan 2004 | 1

Godt beskrevet.

Kommentar af danielhep d. 22. May 2004 | 2

udemærket

Kommentar af conrad d. 29. Feb 2004 | 3

Rigtig god, man kunne måske tilføje lidt om hvordan man stopper en tråd?