MT AG

business by integration

Java Unterstützung für Multithreading von den Versionen 1.0 bis 7

Wolfgang Nast

Nürnberg, 21.11.2013

MT AG

GESCHÄFTSFORM	INHABERGEFÜHRTE AG
HAUPTSITZ	RATINGEN
GRÜNDUNGSJAHR	1994
BESCHÄFTIGTE	180 FESTANGESTELLTE MITARBEITER
BETEILIGUNGEN	MT-IFS GMBH (RATINGEN), MT-IFS SARL (LUXEMBURG)

- BUSINESS INTELLIGENCE SOLUTIONS
- SOCIAL BUSINESS SOLUTIONS
- MOBILE SOLUTIONS
- APPLICATION DEVELOPMENT
- INTEGRATION SERVICES
- IT SYSTEM SERVICES

business by integration

MT AG

Java Unterstützung für Multithreading von den Versionen 1.0 bis 7

- Version 1.0 die Grundlagen
- Version 1.2 die Collections
- Version 1.4 neues Speichermodell
- Version 5 die Concurent Klassen
- Version 6 Erweiterung der Collections
- Version 7 das Fork/Join-Framework
- Version 8 Ausblick auf Neuerungen

Version 1.0 die Grundlagen

MT AG

Grundlagen

- Threads starten und beenden
- Der Status von Threads
- Interaktionen von Threads
- Abbrechen eines Threads(interrupt)
- Synchronisation von Threads
- Die ThreadGroup
- Datenaustausch mit Volatile
- Datenaustausch mit Piped-Stream

Grundlagen

Threads starten und beenden

- Anlegen eines Threads:

```
Thread th = new Thread("ThreadName", runable);
Thread th2 = new ThreadImpl();
```

- Auszuführende Methode:

```
@Override
public void run() {… auszuführender Code …}
```

- Starten mit:

```
th.start();
```

- Beenden mit:

```
return; //in run()
```

Grundlagen

Der Status von Threads

- Abfragen des Status:

```
th.getState();
```

- Die Werte:

```
NEW                 //nicht gestartet
RUNNABLE            //wird ausgeführt
BLOCKED             //geblockt
WAITING             //warten
TIMED_WAITING       //warten mit maximaler Zeit
TERMINATED          //beendet nach Ausführung
```

- Abfragen des aktuellen Threads:

```
Thread.currentThread();
```

Grundlagen

Interaktionen von Threads

- Warten auf beenden eines anderen Threads:

```
th.join();
```

- Warten für eine bestimmte Zeit:

```
Thread.sleep(1000);// eine Sekunde
```

- Thread zu Deamon machen:

```
th.setDeamon(true);
```

- Priorität eines Thread ändern:

```
th.setPriority(Thread.MAX_PRIORITY);
```

- Ausführungsrecht abgeben:

```
Thread.yield();
```

Grundlagen

Abbrechen eines Threads(interrupt)

- Aufforden zum Abbrechen:

```
th.interrupt();
```

- Abbrechen beim Warten oder bei Geblockt:

```
throw InterruptedException();//Thread wird aufgeweckt!
```

- Laufender Thread muss aktiv prüfen, ob er sich beenden soll:

```
if (Thread.interrupted()) // oder
if (th.isInterrupted())
```

Grundlagen

Synchronisation von Threads

- Das Schlüsselwort synchronized:

```
public synchronized void methode()//gegen Instanz
public static synchronized void funkt()//gegen Klasse
sysnchronized(wert){… Block …}//gegen Object "wert"
```

- Im synchronen Block warten:

```
wert.wait();
```

- Im synchronen Block wartenden Thread aufwecken:

```
wert.notify();     //einen wartenden Thread aufwecken.
wert.notifyAll();  //alle wartenden Threads aufwecken.
```

Grundlagen

Die ThreadGroup

- Die ThreadGroup:

```
ThreadGroup tg = new ThreadGroup("GruppenName");
```

- Eine Untergruppe:

```
ThreadGroup subTg = new ThreadGroup(tg, "Sub");
```

- Aktive Threads in Gruppe ermitteln:

```
int anz = tg.activeCount();
Thread thArray[] = new Thread[anz];
tg.enumerate(thArray, true);
```

Grundlagen

•Datenaustausch mit Volatile

- Nur Attribute mit volatile werden nicht in den Threads gecached:

```
private volatile boolean verarbeitet = false;
```

- Austauch eines Wertes:

```
In Thread 1:
verarbeitet = true;
In Thread 2:
if (verarbeitet) {
```

- Vorsicht bei long, int und Referenzen:

```
volatile int anz = 3;
volatile long menge = 4L;
volatile Object verweis = Boolean.TRUE;
```

Grundlagen

•Datenaustausch mit Piped-Stream

- Anlegen des OutputStream:

```
PipedOutputStream po = new PipedOutputStream();
PipedOutputStream po2 = new PipedOutputStream(pi);
```

- Anlegen des InputStream:

```
PipedInputStream pi = new PipedInputStream(po);

PipedInputStream pi2 = new PipedInputStream();
```

- Verbinden der Streams:

```
pi2.connect(po);

po.connect(pi2);
```

•Version 1.2 die Collections

MT AG

Die Collections

- Collections
- ThreadLocal

Die Collections

•Collections

- Die Collections sind nicht mehr synchronized wie:

```
Vector vec = new Vector();
Hashtable map = new Hashtable();
```

- Anlegen von synchronized Collections:

```
Collections.synchronizedCollection(…);
Collections.synchronizedList(…);
Collections.synchronizedMap(…);
```

- Alle Collections bekommen ungültige Iteratoren, wenn sich die Collection ändert:

```
throws ConcurrentModificationException;
```

Die Collections

•ThreadLocal

- Die Thread abhängige Variablen ohne Initialwert:

```
ThreadLocal loc = new ThreadLocal();
```

- Die Thread abhängige Variablen mit erben des Wertes:

```
InheritableThreadLocal loc = new InheritableThreadLocal();
```

- Die Thread abhängige Variablen mit Initialwert:

```
ThreadLocal loc = new ThreadLocal() {
protected synchronized Object initialValue() {
            return …;//initialer Wert
        }
};
```

•Version 1.4 neues Speichermodell

MT AG

Neues Speichermodell

- Das Verhalten von volatile wurde angepasst.
- Neu ist das volatile jetzt verhindert, das die Reihenfolge von Rechenoperation und Zuweisungen nicht mehr vor oder hinter volatile Attribute verschoben werden kann.
- Auch weitere Eigenschaften im Speichermodell wurden an die allgemeinen Erwartungen angepasst.

•Version 5 die Concurent Klassen

MT AG

Die Concurent Klassen

- Locks
- Condition
- Atomic
- Future
- ExecutorService
- Queues, Concurrent Collections
- Latch, Barrier und Exchanger

Die Concurent Klassen

•Locks

- Der einfache Lock:

```
ReentrantLock lock = new ReentrantLock();
try{
lock.lock();
…
}finally{
lock.unlock();
}
```

- Der ReadWriteLock:

```
ReentrantReadWriteLock rwLock = new
ReentrantReadWriteLock();
```

Die Concurent Klassen

•Condition

- Die Condition:

```
lock.lock();
Condition cond = lock.newCondition();
```

- Zum Warten auf einen anderen Thread:

```
cond.await();
```

- Zum Aufwecken eines wartenden Threads:

```
cond.signal();
```

- Zum Aufwecken aller wartenden Threads:

```
cond.signalAll();
```

Die Concurent Klassen

•Atomic

- Die Atomaren Operationen:

```
AtomicBoolean ab;
AtomicInteger ai;
AtomicLong al;
AtomicReference<Object> arO;
AtomicIntegerArray aia;
AtomicLongArray ala;
AtomicReferenceArray<Object> araO;
```

Die Concurent Klassen

•Future

- Das Callable ist ein Runnable mit Rückgabewert:

```
Callable<String> call = new Callable<String>() {
@Override public String call(){
      return "Text";
}};
```

- Ein Future hält ein Callable mit Ergebnis, das ausführbar ist:

```
fut.isDone();      //prüft ob schon ausgeführt.
fut.isCancelled(); //prüft ob abgebrochen.
fut.get();         //holt das Ergebnis
```

Die Concurent Klassen

•ExecutorService

- Der ExecutorService ist zum Ausführen von Calleables mit Future, hier gibt es folgende Standard Implementierungen:

```
ExecutorService ex = Executors.newSingleThreadExecutor();
ex = Executors.newFixedThreadPool(10);
ex = Executors.newCachedThreadPool();
ex = Executors.newScheduledThreadPool(10);
ex = Executors.newSingleThreadScheduledExecutor();
Callable<String> aufr;
Future<String> fut = ex.submit(aufr);
```

Die Concurent Klassen

•Queues, Concurrent Collections

- Der Queues sind geeignet zum Austausch von Werten zwischen Threads:

```
BlockingQueue<String> qu = new LinkedBlockingQueue<>();
```

- Die SynchronousQueue dient zum direkten Austausch zwischen zwei Threads. Hier sind nur put und take erlaubt :

```
SynchronousQueue<String> syqu = new SynchronousQueue<>();
```

- Die Concurend Collections behalten gültige Itteratoren:

```
ConcurrentHashMap
CopyOnWriteArrayList
CopyOnWriteArraySet
```

Die Concurent Klassen

•Latch, Barrier und Exchanger

- Die CountDownLatch wartet mit einer Aktion bis ein andere diese frei gibt:

```
CountDownLatch la = new CountDownLatch(1);
la.await();         //Wartender Thread
la.countDown();     //freigebender Thread
```

- Die CyclicBarrier dient zum synchronisieren von n Threads:

```
CyclicBarrier br = new CyclicBarrier(2);
br.await();         //Thread 1 blockt
br.await();         //Thread 2, beide Threads laufen wieder
```

- Der Exchanger dient zum Austausch von Werten:

```
Exchanger<String> ex = new Exchanger<>();
String a = ex.exchange("Wert1");
String b = ex.exchange("Wert2");
```

•Version 6 Erweiterung der Collections

MT AG

Erweiterung der Collections

- Die Deque erweitert die Queues:

```
BlockingDeque                 //Interface
LinkedBlockingDeque           //Implementierung
```

- Die neuen Concurend Collections mit gültigen Iteratoren:

```
ConcurrentSkipListMap
ConcurrentSkipListSet
```

•Version 7 das Fork/Join-Framework

MT AG

•Das Fork/Join-Framework

- Fork/Join
- LinkedTransferQueue
- ConcurrentLinkedDeque
- Phaser
- ThreadLocalRandom

Das Fork/Join-Framework

•Fork/Join

- Der neue ForkJoinTask erweitert Future:

```
ForkJoinTask<String> fjTask = ForkJoinTask.adapt(callable);
```

- Als Spezialisierung von ForkJoinTask gibt es den RecursiveTask:

```
RecursiveTask<String> rcTask;// Methode compute() ist zu
//überschreiben
```

- Der ForkJoinPool ist zum Ausführen der Tasks:

```
ForkJoinPool pool = ForkJoinTask.getPool();
```

Das Fork/Join-Framework

•LinkedTransferQueue, ConcurrentLinkedDeque

- Die LinkedTransferQueue ermöglicht den direkten Transfer an den andern Thread:

```
LinkedTransferQueue<String> tq = new
LinkedTransferQueue<String>();

tq.transfer("B");
```

- Die ConcurrentLinkedDeque eignet sich für mehr als zwei Threads, die sich eine Liste teilen:

```
ConcurrentLinkedDeque<String> ld = new
ConcurrentLinkedDeque<>();
```

•Version 8 Ausblick auf Neuerungen

MT AG

•Ausblick auf Neuerungen

- Streams, sorted und parallel
- Neuer Lock

Fragen ?

MT AG

Vorträge auf der DOAG 2013

ADF Persistenz-Frameworks im Vergleich – JPA/EJB vs. ADF BC	Hendrik Gossens, Di, 11 Uhr
Continuous Integration für Oracle DB und Apex	Peter Busch, Dominic Ketteltasche, Di, 12 Uhr
Mein Backup – die richtige Strategie oder de Irrweg?	Volker Mach, Di 16 Uhr
Das APEX QS-Plugin	Oliver Lemm, Mi, 12 Uhr
Forms goes APEX – wie man es ri	Niels de Bruijn, Sven-Olaf Kelbert, Mi 15 Uhr
USABLE_FILE_MB im Ora m der nutzbare Plattenplatz ne	Ernst Leber, Mi, 16 Uhr
Erstellen einer mobilen Ap mit PhoneGap und ADF Mobile	Wolfgang Nast, Do, 9 Uhr
Experten-Panel: APEX und DB-Programmierung	Niels de Bruijn, Do, 11 Uhr
Java Unterstützung von Multithreading in den Versionen 1.0 bis 7	Wolfgang Nast, Do, 12 Uhr
3 Wochenenden Strohwitwer	Christof Kaller, Do, 12 Uhr

Stand 328

business by integration

MT AG

MT AG

business by integration

Vielen Dank.

MT AG

Balcke-Dürr-Allee 9
40882 Ratingen

Telefon: +49 (0) 21 02 309 61-0
Telefax: +49 (0) 21 02 309 61-10

E-Mail: info@mt-ag.com
www.mt-ag.com