Dataflow

The Whole GPars Team <gpars-developers@googlegroups.com>

Version 1.2.1, 2015-12-01

Table of Contents

Dataflow

`Dataflow concurrency` offers an alternative concurrency model, which is inherently safe and robust.

Introduction

Check out the small example written in **Groovy** using **GPars**, which sums results of calculations performed by three concurrently run tasks:

Dataflow Sample

```
import static groovyx.gpars.dataflow.Dataflow.task
final def x = new DataflowVariable()
final def y = new DataflowVariable()
final def z = new DataflowVariable()

task {
    z << x.val + y.val
    println "Result: ${z.val}"
}

task {
    x << 10
}

task {
    y << 5
}
```

Review

We start three logical tasks, which can run in parallel and perform their particular activities. The tasks need to exchange data and they do so using *Dataflow Variables*. Think of *Dataflow Variables* as one-shot channels safely and reliably tranferring data from producers to their consumers.

The *Dataflow Variables* have a pretty straightforward semantics. When a task needs to read a value from *DataflowVariable* (through the val property), it will block until the value has been set by another task or thread (using the '<<' operator). Each *DataflowVariable* can be set **only once** in its lifetime. Notice that you don't have to bother with ordering and synchronizing the tasks or threads and their access to shared variables. The values are magically transferred among tasks at the right time without your intervention.

The data flows seamlessly among tasks / threads without your intervention or care.

Implementation detail

The three tasks in the previous example **do not necessarily need to be mapped to three physical threads**. Tasks represent so-called "green" or "logical" threads and can be mapped under the covers to any number of physical threads.

The actual mapping depends on the scheduler, but the outcome of dataflow algorithms doesn't depend on the actual scheduling.

Benefits

Here's what you gain by using Dataflow Concurrency (by Jonas Boner www.jonasboner.com):

- No race-conditions
- No live-locks
- Deterministic deadlocks
- Completely deterministic programs
- *BEAUTIFUL* code.

This doesn't sound bad, does it?

If you'd like to know more on this interesting concept, check out the Dataflow concurrency section of the User Guide.

Further reading

- Scala Dataflow library by Jonas Bonner
- JVM concurrency presentation slides by Jonas Bonner
- Dataflow Concurrency library for Ruby