Fork/Join

Russell Winder

Version 1.0, 2015-10-01

Table of Contents

Concepts

Fork / Join

Fork/Join, or *Divide and Conquer*, is a very powerful abstraction to solve hierarchical problems. When talking about hierarchical problems, think about quick sort, merge sort, file system or general tree navigation and such.

- **Fork / Join** algorithms essentially split a problem at hands into several smaller sub-problems and recursively apply the same algorithm to each of the sub-problems.
- Once the sub-problem is small enough, it is solved directly.
- The solutions of all sub-problems are combined to solve their parent problem, which in turn helps solve its own parent problem.

Usage

Using the Fork-Join Builder

Feel free to experiment with the number of fork/join threads in the pool

A Sample

```
withPool(1){pool ->

    println """Number of files: ${

        runForkJoin(new File("./src")) {file ->
            long count = 0

            file.eachFile {
                if (it.isDirectory()) {
                    println "Forking a child task for $it"
                    // Fork a child task.
                    forkOffChild(it)
                } else {
                    count++
                }
            }

            // Use results of children tasks to calculate and store own result.
            return count + (childrenResults.sum(0))
        }
    }"""
}
```

Extending the *AbstractForkJoinWorker* class

```
public final class FileCounter extends AbstractForkJoinWorker<Long> {
    private final File file;

    def FileCounter(final File file) {
        this.file = file
    }

    protected void compute() {
        long count = 0;
        file.eachFile{
            if (it.isDirectory()) {
                println "Forking a thread for $it"
                // Fork a child task.
                forkOffChild(new FileCounter(it))
            }
            else {
                count++
            }
        }

        // Use results of children tasks to calculate and store own result.
        setResult(count + ((childrenResults)?.sum() ?: 0))
    }
}

withPool(1){pool ->  // Feel free to experiment with the number of fork/join threads in
the pool.
    println "Number of files: ${orchestrate(new FileCounter(new File("..")))}"
}
```