Java 核心技术(进阶)

第五章 Java多线程和并发编程

第六节 Java并发框架Fork-Join

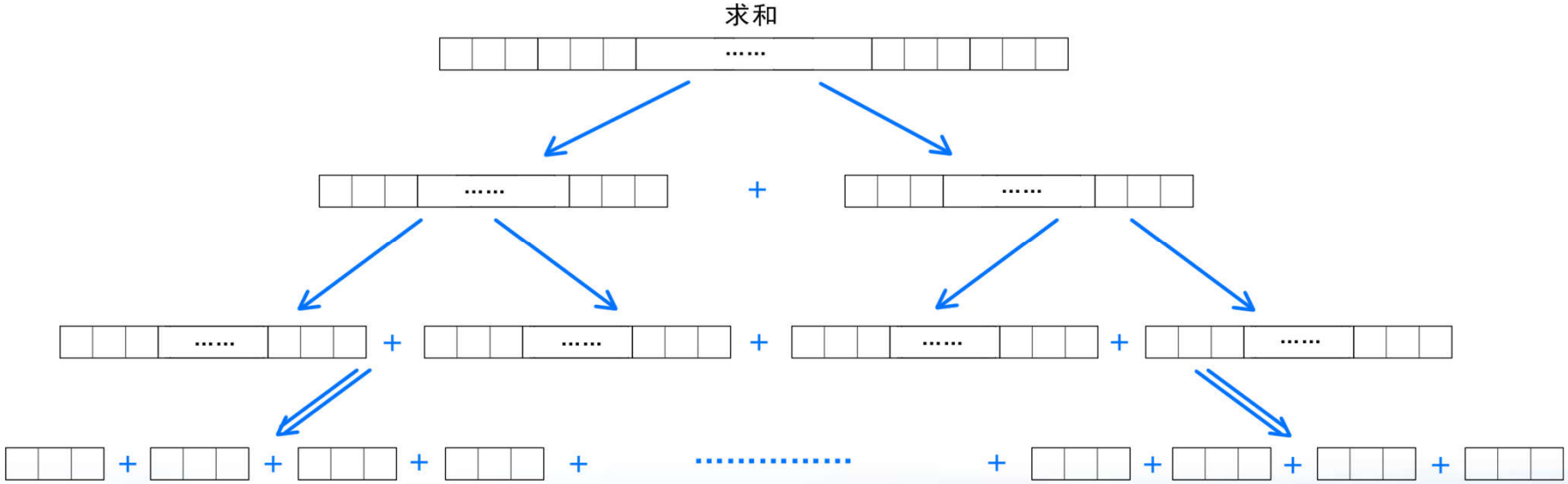
华东师范大学 陈良育

并行计算

- 并行模式
  - 主从模式 (Master-Slave)
  - Worker模式(Worker-Worker)
- Java并发编程
  - Thread/Runnable/Thread组管理
  - Executor
  - Fork-Join框架(本节重点)

Fork-Join(1)

- Java 7 提供另一种并行框架：分解、治理、合并(分治编程)
- 适合用于整体任务量不好确定的场合(最小任务可确定)



Fork-Join(2)

- 关键类
  - ForkJoinPool 任务池
  - RecursiveAction
  - RecursiveTask
- 参看例子

总结

- 了解Fork-Join和Executor的区别
- 控制任务分解粒度
- 熟悉Fork-join框架，提高多线程执行效率

代码(1) SumTest.java


```
//分任务求和
public class SumTest {

    public static void main(String[] args) throws ExecutionException, InterruptedException {
        //创建执行线程池
        ForkJoinPool pool = new ForkJoinPool();
        //ForkJoinPool pool = new ForkJoinPool(4);

        //创建任务
        SumTask task = new SumTask(1, 10000000);

        //提交任务
        ForkJoinTask<Long> result = pool.submit(task);
```

代码(2) SumTest.java


```
        //等待结果
        do {
            System.out.printf("Main: Thread Count: %d\n",pool.getActiveThreadCount());
            System.out.printf("Main: Paralelism: %d\n",pool.getParallelism());
            try {
                Thread.sleep(50);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        } while (!task.isDone());

        //输出结果
        System.out.println(result.get().toString());
    }
}
```

代码(3) SumTask.java


```
//分任务求和
public class SumTask extends RecursiveTask<Long> {

    private int start;
    private int end;

    public SumTask(int start, int end) {
        this.start = start;
        this.end = end;
    }

    public static final int threadhold = 5;
```

代码(4) SumTask.java


```
    @Override
    protected Long compute() {
        Long sum = 0L;

        // 如果任务足够小，就直接执行
        boolean canCompute = (end - start) <= threadhold;
        if (canCompute) {
            for (int i = start; i <= end; i++) {
                sum = sum + i;
            }
        } else {
            // 任务大于阈值，分裂为2个任务
            int middle = (start + end) / 2;
            SumTask subTask1 = new SumTask(start, middle);
            SumTask subTask2 = new SumTask(middle + 1, end);

            invokeAll(subTask1, subTask2);

            Long sum1 = subTask1.join();
            Long sum2 = subTask2.join();

            // 结果合并
            sum = sum1 + sum2;
        }
        return sum;
    }
}
```

谢 谢！