Java 核心技术(进阶)

第五章 Java多线程和并发编程

第五节 Java并发框架Executor

华东师范大学 陈良育

并行计算(1)

- 业务：任务多，数据量大
- 串行 vs 并行
  - 串行编程简单，并行编程困难
  - 单个计算核频率下降，计算核数增多，整体性能变高
- 并行困难(任务分配和执行过程高度耦合)
  - 如何控制粒度，切割任务
  - 如何分配任务给线程，监督线程执行过程

并行计算(2)

- 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(本节重点)
  - Fork-Join框架

线程组管理

- 线程组ThreadGroup
  - 线程的集合
  - 树形结构，大线程组可以包括小线程组
  - 可以通过enumerate方法遍历组内的线程，执行操作
  - 能够有效管理多个线程，但是管理效率低
  - 任务分配和执行过程高度耦合
  - 重复创建线程、关闭线程操作，无法重用线程
- 参看例子

Executor(1)

- 从JDK 5开始提供Executor FrameWork (java.util.concurrent.*)
  - 分离任务的创建和执行者的创建
  - 线程重复利用(new线程代价很大)
- 理解共享线程池的概念
  - 预设好的多个Thread，可弹性增加
  - 多次执行很多很小的任务
  - 任务创建和执行过程解耦
  - 程序员无需关心线程池执行任务过程

Executor(2)

- 主要类：ExecutorService, ThreadPoolExecutor，Future
  - Executors.newCachedThreadPool/newFixedThreadPool 创建线程池
  - ExecutorService 线程池服务
  - Callable 具体的逻辑对象(线程类)
  - Future 返回结果
- 参看例子

总结

- 掌握共享线程池原理
- 熟悉Executor框架，提高多线程执行效率

代码(1) Main.java


```
public class Main {

    public static void main(String[] args) {

        // 创建线程组
        ThreadGroup threadGroup = new ThreadGroup("Searcher");
        Result result=new Result();

        // 创建一个任务，10个线程完成
        Searcher searchTask=new Searcher(result);
        for (int i=0; i<10; i++) {
            Thread thread=new Thread(threadGroup, searchTask);
            thread.start();
            try {
                TimeUnit.SECONDS.sleep(1);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        System.out.println("========华丽丽0=======");
```

代码(2) Main.java


```
// 查看线程组消息
System.out.printf("active 线程数量: %d\n",threadGroup.activeCount());
System.out.printf("线程组信息明细\n");
threadGroup.list();
System.out.println("========华丽丽1=======");

// 遍历线程组
Thread[] threads=new Thread[threadGroup.activeCount()];
threadGroup.enumerate(threads);
for (int i=0; i<threadGroup.activeCount(); i++) {
    System.out.printf("Thread %s: %s\n",threads[i].getName(),threads[i].getState());
}
System.out.println("========华丽丽2=======");
```

代码(3) Main.java


```
        // Wait for the finalization of the Threadds
        waitFinish(threadGroup);

        // Interrupt all the Thread objects assigned to the ThreadGroup
        threadGroup.interrupt();
    }

    public static void waitFinish(ThreadGroup threadGroup) {
        while (threadGroup.activeCount()>9) {
            try {
                TimeUnit.SECONDS.sleep(1);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

代码(4) Result.java


```
public class Result {

    private String name;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }

}
```

代码(5) Searcher.java


```
public class Searcher implements Runnable {

    private Result result;

    public Searcher(Result result) {
        this.result=result;
    }

    @Override
    public void run() {
        String name=Thread.currentThread().getName();
        System.out.printf("Thread %s: 启动\n",name);
        try {
            doTask();
            result.setName(name);
        } catch (InterruptedException e) {
            System.out.printf("Thread %s: 被中断\n",name);
            return;
        }
        System.out.printf("Thread %s: 完成\n",name);
    }
```

代码(6) Searcher.java


```
    private void doTask() throws InterruptedException {
        Random random=new Random((new Date()).getTime());
        int value=(int)(random.nextDouble()*100);
        System.out.printf("Thread %s: %d\n",Thread.currentThread().getName(),value);
        TimeUnit.SECONDS.sleep(value);
    }
}
```

代码(7) Main.java


```
package executor.example1;

public class Main {

    public static void main(String[] args) throws InterruptedException {
        // 创建一个执行服务器
        Server server=new Server();

        // 创建100个任务，并发给执行器，等待完成
        for (int i=0; i<100; i++){
            Task task=new Task("Task "+i);
            Thread.sleep(10);
            server.submitTask(task);
        }
        server.endServer();
    }
}
```

代码(8) Server.java

```
public class Server {

    //线程池
    private ThreadPoolExecutor executor;

    public Server(){
        executor=(ThreadPoolExecutor)Executors.newCachedThreadPool();
        //executor=(ThreadPoolExecutor)Executors.newFixedThreadPool(5);
    }

    //向线程池提交任务
    public void submitTask(Task task){
        System.out.printf("Server: A new task has arrived\n");
        executor.execute(task); //执行 无返回值

        System.out.printf("Server: Pool Size: %d\n",executor.getPoolSize());
        System.out.printf("Server: Active Count: %d\n",executor.getActiveCount());
        System.out.printf("Server: Completed Tasks: %d\n",executor.getCompletedTaskCount());
    }

    public void endServer() {
        executor.shutdown();
    }
}
```

代码(9) Task.java


```java
public class Task implements Runnable {

    private String name;

    public Task(String name){
        this.name=name;
    }

    public void run() {
        try {
            Long duration=(long)(Math.random()*1000);
            System.out.printf("%s: Task %s: Doing a task during %d seconds\n",Thread.currentThread().getName(),name,duration);
            Thread.sleep(duration);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        System.out.printf("%s: Task %s: Finished on: %s\n",Thread.currentThread().getName(),name,new Date());
    }

}
```

代码(10) SumTest.java


```
public class SumTest {

    public static void main(String[] args) {

        // 执行线程池
        ThreadPoolExecutor executor=(ThreadPoolExecutor)Executors.newFixedThreadPool(4);

        List<Future<Integer>> resultList=new ArrayList<>();

        //统计1-1000总和，分成10个任务计算，提交任务
        for (int i=0; i<10; i++){
            SumTask calculator=new SumTask(i*100+1, (i+1)*100);
            Future<Integer> result=executor.submit(calculator);
            resultList.add(result);
        }
```

代码(11) SumTest.java


```
// 每隔50毫秒，轮询等待10个任务结束
do {
    System.out.printf("Main: 已经完成多少个任务: %d\n",executor.getCompletedTaskCount());
    for (int i=0; i<resultList.size(); i++) {
        Future<Integer> result=resultList.get(i);
        System.out.printf("Main: Task %d: %s\n",i,result.isDone());
    }
    try {
        Thread.sleep(50);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
} while (executor.getCompletedTaskCount()<resultList.size());
```

代码(12) SumTest.java


```
        // 所有任务都已经结束了，综合计算结果
        int total = 0;
        for (int i=0; i<resultList.size(); i++) {
            Future<Integer> result=resultList.get(i);
            Integer sum=null;
            try {
                sum=result.get();
                total = total + sum;
            } catch (InterruptedException e) {
                e.printStackTrace();
            } catch (ExecutionException e) {
                e.printStackTrace();
            }
        }
        System.out.printf("1-1000的总和:" + total);

        // 关闭线程池
        executor.shutdown();
    }
}
```

代码(13) SumTask.java


```
public class SumTask implements Callable<Integer> {
    //定义每个线程计算的区间
    private int startNumber;
    private int endNumber;

    public SumTask(int startNumber, int endNumber){
        this.startNumber=startNumber;
        this.endNumber=endNumber;
    }

    @Override
    public Integer call() throws Exception {
        int sum = 0;
        for(int i=startNumber; i<=endNumber; i++)
        {
            sum = sum + i;
        }

        Thread.sleep(new Random().nextInt(1000));

        System.out.printf("%s: %d\n",Thread.currentThread().getName(),sum);
        return sum;
    }
}
```

谢 谢！