链滴

java 多线程分批处理数据工具类

作者：zytops

原文链接：https://ld246.com/article/1570774213067

来源网站：链滴

数据量比较大，需要使用多线程来分批处理，提高处理效率和能力，于是就写了一个通用的多线程处工具，只需要实现自己的业务逻辑就可以正常使用，现在记录一下

主要是针对大数据量list，将list划分多个线程处理

ResultBean类： 返回结果统一bean

```
package com.ts.common.model;

import java.io.Serializable;

import com.alibaba.fastjson.JSON;


/**
 * 返回结果统一bean
 *
 * ResultBean<BR>
 * 创建人:wangbeidou <BR>
 * 时间：2018年4月12日-下午3:49:46 <BR>
 * @version 2.0
 *
 */
public class ResultBean<T> implements Serializable {

    private static final long serialVersionUID = 1L;
    // 成功状态
    public static final int SUCCESS = 1;
    // 处理中状态
    public static final int PROCESSING = 0;
    // 失败状态
    public static final int FAIL = -1;

    // 描述
    private String msg = "success";
    // 状态默认成功
    private int code = SUCCESS;
    // 备注
    private String remark;
    // 返回数据
    private T data;

    public ResultBean() {
        super();
    }

    public ResultBean(T data) {
        super();
        this.data = data;
    }

    /**
     * 使用异常创建结果
     */
```

```
public ResultBean(Throwable e) {
    super();
    this.msg = e.toString();
    this.code = FAIL;
}

/**
 *
 * 实例化结果默认成功状态<BR>
 * 方法名：newInstance<BR>
 * 创建人：wangbeidou <BR>
 * 时间：2018年4月12日-下午3:51:26 <BR>
 * @return ResultBean<T><BR>
 * @exception <BR>
 * @since  2.0
 */
public static <T> ResultBean<T> newInstance() {
    ResultBean<T> instance = new ResultBean<T>();
    //默认返回信息
    instance.code = SUCCESS;
    instance.msg = "success";
    return instance;
}

/**
 *
 * 实例化结果默认成功状态和数据<BR>
 * 方法名：newInstance<BR>
 * 创建人：wangbeidou <BR>
 * 时间：2018年5月10日-下午2:13:16 <BR>
 * @param data
 * @return ResultBean<T><BR>
 * @exception <BR>
 * @since  2.0
 */
public static <T> ResultBean<T> newInstance(T data) {
    ResultBean<T> instance = new ResultBean<T>();
    //默认返回信息
    instance.code = SUCCESS;
    instance.msg = "success";
    instance.data = data;
    return instance;
}

/**
 *
 * 实例化返回结果<BR>
 * 方法名：newInstance<BR>
 * 创建人：wangbeidou <BR>
 * 时间：2018年4月12日-下午4:00:53 <BR>
 * @param code
 * @param msg
 * @return ResultBean<T><BR>
 * @exception <BR>
```

```
     * @since  2.0
     */
    public static <T> ResultBean<T> newInstance(int code, String msg) {
        ResultBean<T> instance = new ResultBean<T>();
        //默认返回信息
        instance.code = code;
        instance.msg = msg;
        return instance;
    }

    /**
     *
     * 实例化返回结果<BR>
     * 方法名：newInstance<BR>
     * 创建人：wangbeidou <BR>
     * 时间：2018年4月12日-下午4:00:35 <BR>
     * @param code
     * @param msg
     * @param data
     * @return ResultBean<T><BR>
     * @exception <BR>
     * @since  2.0
     */
    public static <T> ResultBean<T> newInstance(int code, String msg, T data) {
        ResultBean<T> instance = new ResultBean<T>();
        //默认返回信息
        instance.code = code;
        instance.msg = msg;
        instance.data = data;
        return instance;
    }

    /**
     *
     * 设置返回数据<BR>
     * 方法名：setData<BR>
     * 创建人：wangbeidou <BR>
     * 时间：2018年4月12日-下午3:52:01 <BR>
     * @param data
     * @return ResultBean<T><BR>
     * @exception <BR>
     * @since  2.0
     */
    public ResultBean<T> setData(T data){
        this.data = data;
        return this;
    }

    /**
     *
     * 设置结果描述<BR>
     * 方法名：setMsg<BR>
     * 创建人：wangbeidou <BR>
     * 时间：2018年4月12日-下午3:52:34 <BR>
```

```
 * @param msg
 * @return ResultBean<T><BR>
 * @exception <BR>
 * @since  2.0
 */
public ResultBean<T> setMsg(String msg){
    this.msg = msg;
    return this;
}

/**
 *
 * 设置状态<BR>
 * 方法名：setCode<BR>
 * 创建人：wangbeidou <BR>
 * 时间：2018年4月12日-下午4:17:56 <BR>
 * @param code
 * @return ResultBean<T><BR>
 * @exception <BR>
 * @since  2.0
 */
public ResultBean<T> setCode(int code){
    this.code = code;
    return this;
}

/**
 *
 * 设置备注)<BR>
 * 方法名：setRemark<BR>
 * 创建人：wangbeidou <BR>
 * 时间：2018年4月12日-下午5:47:29 <BR>
 * @param remark
 * @return ResultBean<T><BR>
 * @exception <BR>
 * @since  2.0
 */
public ResultBean<T> setRemark(String remark){
    this.remark = remark;
    return this;
}

/**
 *
 * 设置成功描述和返回数据<BR>
 * 方法名：success<BR>
 * 创建人：wangbeidou <BR>
 * 时间：2018年4月12日-下午3:52:58 <BR>
 * @param msg
 * @param data
 * @return ResultBean<T><BR>
 * @exception <BR>
 * @since  2.0
 */
```

```
public ResultBean<T> success(String msg, T data){
    this.code = SUCCESS;
    this.data = data;
    this.msg = msg;
    return this;
}

/**
 *
 * 设置成功返回结果描述<BR>
 * 方法名：success<BR>
 * 创建人：wangbeidou <BR>
 * 时间：2018年4月12日-下午3:53:31 <BR>
 * @param msg
 * @return ResultBean<T><BR>
 * @exception <BR>
 * @since  2.0
 */
public ResultBean<T> success(String msg){
    this.code = SUCCESS;
    this.msg = msg;
    return this;
}

/**
 *
 * 设置处理中描述和返回数据<BR>
 * 方法名：success<BR>
 * 创建人：wangbeidou <BR>
 * 时间：2018年4月12日-下午3:52:58 <BR>
 * @param msg
 * @param data
 * @return ResultBean<T><BR>
 * @exception <BR>
 * @since  2.0
 */
public ResultBean<T> processing(String msg, T data){
    this.code = PROCESSING;
    this.data = data;
    this.msg = msg;
    return this;
}

/**
 *
 * 设置处理中返回结果描述<BR>
 * 方法名：success<BR>
 * 创建人：wangbeidou <BR>
 * 时间：2018年4月12日-下午3:53:31 <BR>
 * @param msg
 * @return ResultBean<T><BR>
 * @exception <BR>
 * @since  2.0
 */
```

```
public ResultBean<T> processing(String msg){
    this.code = PROCESSING;
    this.msg = msg;
    return this;
}

/**
 *
 * 设置失败返回描述和返回数据<BR>
 * 方法名：fail<BR>
 * 创建人：wangbeidou <BR>
 * 时间：2018年4月12日-下午3:54:04 <BR>
 * @param msg
 * @param data
 * @return ResultBean<T><BR>
 * @exception <BR>
 * @since  2.0
 */
public ResultBean<T> fail(String msg, T data){
    this.code = FAIL;
    this.data = data;
    this.msg = msg;
    return this;
}

/**
 *
 * 设置失败返回描述<BR>
 * 方法名：fail<BR>
 * 创建人：wangbeidou <BR>
 * 时间：2018年4月12日-下午3:54:32 <BR>
 * @param msg
 * @return ResultBean<T><BR>
 * @exception <BR>
 * @since  2.0
 */
public ResultBean<T> fail(String msg){
    this.code = FAIL;
    this.msg = msg;
    return this;
}

public T getData() {
    return data;
}
public String getMsg() {
    return msg;
}
public int getCode() {
    return code;
}
public String getRemark() {
    return remark;
}
```

```
    /**
     *
     * 生成json字符串<BR>
     * 方法名：json<BR>
     * 创建人：wangbeidou <BR>
     * 时间：2018年4月12日-下午4:42:28 <BR>
     * @return String<BR>
     * @exception <BR>
     * @since  2.0
     */
    public String json(){
        return JSON.toJSONString(this);
    }
}
```

ITask接口： 实现自己的业务

```
package com.ts.common.multi.execute;

import java.util.Map;

/**
 * 任务处理接口
 * 具体业务逻辑可实现该接口
 *  T 返回值类型
 *  E 传入值类型
 * ITask<BR>
 * 创建人:wangbeidou <BR>
 * 时间：2018年8月4日-下午6:12:32 <BR>
 * @version 2.0
 *
 */
public interface ITask<T, E> {

    /**
     *
     * 任务执行方法接口<BR>
     * 方法名：execute<BR>
     * 创建人：wangbeidou <BR>
     * 时间：2018年8月4日-下午6:13:44 <BR>
     * @param e 传入对象
     * @param params 其他辅助参数
     * @return T<BR> 返回值类型
     * @exception <BR>
     * @since  2.0
     */
    T execute(E e, Map<String, Object> params);
}
```

HandleCallable类： 实现Callable接口，来处理任务

```
package com.ts.common.multi.execute;
```

```
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.util.concurrent.Callable;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.ts.common.model.ResultBean;

/**
 *
 *
 * HandleCallable<BR>
 * 创建人:wangbeidou <BR>
 * 时间：2018年8月4日-上午11:55:41 <BR>
 *
 * @version 2.0
 *
 */
@SuppressWarnings("rawtypes")
public class HandleCallable<E> implements Callable<ResultBean> {
    private static Logger logger = LoggerFactory.getLogger(HandleCallable.class);
    // 线程名称
    private String threadName = "";
    // 需要处理的数据
    private List<E> data;
    // 辅助参数
    private Map<String, Object> params;
    // 具体执行任务
    private ITask<ResultBean<String>, E> task;

    public HandleCallable(String threadName, List<E> data, Map<String, Object> params,
            ITask<ResultBean<String>, E> task) {
        this.threadName = threadName;
        this.data = data;
        this.params = params;
        this.task = task;
    }

    @Override
    public ResultBean<List<ResultBean<String>>> call() throws Exception {
        // 该线程中所有数据处理返回结果
        ResultBean<List<ResultBean<String>>> resultBean = ResultBean.newInstance();
        if (data != null && data.size() > 0) {
            logger.info("线程：{},共处理:{}个数据，开始处理.....", threadName, data.size());
            // 返回结果集
            List<ResultBean<String>> resultList = new ArrayList<>();
            // 循环处理每个数据
            for (int i = 0; i < data.size(); i++) {
                // 需要执行的数据
                E e = data.get(i);
                // 将数据执行结果加入到结果集中
                resultList.add(task.execute(e, params));
```

```
                logger.info("线程：{},第{}个数据，处理完成", threadName, (i + 1));
            }
            logger.info("线程：{},共处理:{}个数据，处理完成.....", threadName, data.size());
            resultBean.setData(resultList);
        }
        return resultBean;
    }

}
```

MultiThreadUtils类：多线程工具类

```
package com.ts.common.multi.execute;

import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.util.concurrent.CompletionService;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.ExecutorCompletionService;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.ts.common.model.ResultBean;

/**
 *
 *
 * MultiThreadUtils<BR>
 * 创建人:wangbeidou <BR>
 * 时间：2018年8月8日-下午8:20:42 <BR>
 * @version 2.0
 *
 */
public class MultiThreadUtils<T> {
    private static Logger logger = LoggerFactory.getLogger(MultiThreadUtils.class);

    // 线程个数，如不赋值，默认为5
    private int threadCount = 5;
    // 具体业务任务
    private ITask<ResultBean<String>, T> task;
    // 线程池管理器
    private CompletionService<ResultBean> pool = null;

    /**
     *
     * 初始化线程池和线程个数<BR>
     * 方法名：newInstance<BR>
     * 创建人：wangbeidou <BR>
```

```
     * 时间：2018年8月8日-下午8:22:00 <BR>
     * @param threadCount
     * @return MultiThreadUtils<BR>
     * @exception <BR>
     * @since  2.0
     */
    public static MultiThreadUtils newInstance(int threadCount) {
        MultiThreadUtils instance = new MultiThreadUtils();
        threadCount = threadCount;
        instance.setThreadCount(threadCount);

        return instance;
    }

    /**
     *
     * 多线程分批执行list中的任务<BR>
     * 方法名：execute<BR>
     * 创建人：wangbeidou <BR>
     * 时间：2018年8月8日-下午8:22:31 <BR>
     * @param data  线程处理的大数据量list
     * @param params    处理数据是辅助参数传递
     * @param task      具体执行业务的任务接口
     * @return ResultBean<BR>
     * @exception <BR>
     * @since  2.0
     */
    @SuppressWarnings("rawtypes")
    public ResultBean execute(List<T> data, Map<String, Object> params, ITask<ResultBean<S
ring>, T> task) {
        // 创建线程池
        ExecutorService threadpool = Executors.newFixedThreadPool(threadCount);
        // 根据线程池初始化线程池管理器
        pool = new ExecutorCompletionService<ResultBean>(threadpool);
        // 开始时间（ms）
        long l = System.currentTimeMillis();
        // 数据量大小
        int length = data.size();
        // 每个线程处理的数据个数
        int taskCount = length / threadCount;
        // 划分每个线程调用的数据
        for (int i = 0; i < threadCount; i++) {
            // 每个线程任务数据list
            List<T> subData = null;
            if (i == (threadCount - 1)) {
                subData = data.subList(i * taskCount, length);
            } else {
                subData = data.subList(i * taskCount, (i + 1) * taskCount);
            }
            // 将数据分配给各个线程
            HandleCallable execute = new HandleCallable<T>(String.valueOf(i), subData, params,
ask);
            // 将线程加入到线程池
            pool.submit(execute);
```

```
        }

        // 总的返回结果集
        List<ResultBean<String>> result = new ArrayList<>();
        for (int i = 0; i < threadCount; i++) {
            // 每个线程处理结果集
            ResultBean<List<ResultBean<String>>> threadResult;
            try {
                threadResult = pool.take().get();
                result.addAll(threadResult.getData());
            } catch (InterruptedException e) {
                e.printStackTrace();
            } catch (ExecutionException e) {
                e.printStackTrace();
            }

        }
        // 关闭线程池
        threadpool.shutdownNow();
        // 执行结束时间
        long end_l = System.currentTimeMillis();
        logger.info("总耗时:{}ms", (end_l - l));
        return ResultBean.newInstance().setData(result);
    }

    public int getThreadCount() {
        return threadCount;
    }

    public void setThreadCount(int threadCount) {
        this.threadCount = threadCount;
    }

}
```

测试类TestTask

```
package com.ts.common.multi.execute;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import com.ts.common.model.ResultBean;

/**
 *
 * 具体执行业务任务 需要 实现ITask接口  在execute中重写业务逻辑
 * TestTask<BR>
 * 创建人:wangbeidou <BR>
 * 时间：2018年8月8日-下午8:40:32 <BR>
 * @version 2.0
 *
```

```
 */
public class TestTask implements ITask<ResultBean<String>, Integer> {

    @Override
    public ResultBean execute(Integer e, Map<String, Object> params) {
        /**
         * 具体业务逻辑：将list中的元素加上辅助参数中的数据返回
         */
        int addNum = Integer.valueOf(String.valueOf(params.get("addNum")));
        e = e + addNum;
        ResultBean<String> resultBean = ResultBean.newInstance();
        resultBean.setData(e.toString());
        return resultBean;
    }

    public static void main(String[] args) {
        // 需要多线程处理的大量数据list
        List<Integer> data = new ArrayList<>(10000);
        for(int i = 0; i < 10000; i ++){
            data.add(i + 1);
        }

        // 创建多线程处理任务
        MultiThreadUtils<Integer> threadUtils = MultiThreadUtils.newInstance(5);
        ITask<ResultBean<String>, Integer> task = new TestTask();
        // 辅助参数  加数
        Map<String, Object> params = new HashMap<>();
        params.put("addNum", 4);
        // 执行多线程处理，并返回处理结果
        ResultBean<List<ResultBean<String>>> resultBean = threadUtils.execute(data, params,
ask);
    }
}
```