# Spring Batch

Spring Batch是轻量级、完善的批处理框架，旨在帮助企业建立健壮、高效的批处理框架，用于企业级大批量数据处理应用。Spring batch不仅提供了统一的读写接口、丰富的任务处理方式、灵活的事务管理及并发处理，同时还支持包括日志记录/追踪、事务管理、job处理分析、job流程管理及资源分配等，大大简化了批处理应用开发，将开发人员从复杂的任务配置过程中解放处理，使他们更多的去关注核心的业务处理过程。

# Spring Batch概念及执行

批处理定期读取大批量数据，完成相应业务处理并进行归档，其具有以下几个特点：

* 数据量大，少则百万，多则上亿的数量级
* 不需要人工干预，由系统根据配置自动完成
* 与时间相关，如每天执行一次或每月执行一次

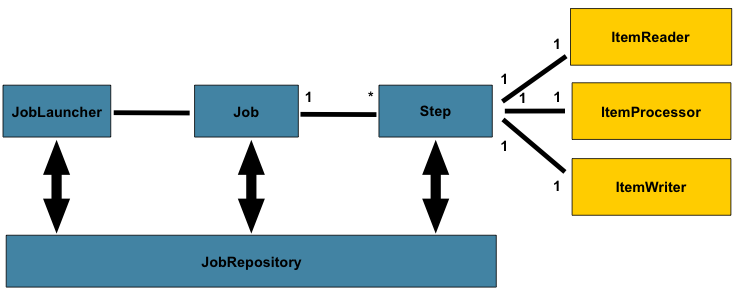
同时，批处理又明显分为三个环节：

* 读数据，数据可能来自文件、数据库或者消息队列等
* 数据处理，如电信支撑系统的计费处理
* 写数据，将输出结果写入文件、数据库或消息队列等

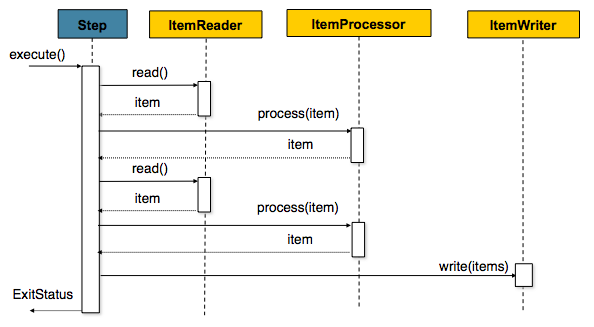
从系统架构上，应重点考虑批处理应用的事务粒度、日志监控、执行、资源管理等，从设计上看，应重点考虑数据读写与业务处理的解耦，提高复用性及可测试性。Spring Batch针对以上环节，由以下几部分组成：



其执行流程图如下所示：



每一个Spring Batch都包含一个Job（类比容器），Job中装配了若干Step，这些Step包括读读数据、处理数据、然后将这些数据存储起来。JobLauncher用于启动Job，JobRepository提供持久化机制。外部控制器调用JobLauncher启动Job，执行序列图：



从DB或者文件中读取数据时，read操作每次只读取一条记录，然后将这条数据传递给processor处理，框架重复以上操作，直到读取的件数超过batch配置的提交设定值，然后进行一次write操作。重复上图的处理，直到处理完所有数据。

http://www.cnblogs.com/gulvzhe/archive/2011/10/25/2224249.html

# Spring Batch的使用

下面通过Spring Batch框架实现一个简单的功能

1. 引入依赖，在pom.xml文件中加入以下代码

*<dependency>*

*<groupId>org.springframework.batch</groupId>*

*<artifactId>spring-batch-core</artifactId>*

*<version>3.0.8.RELEASE</version>*

*</dependency>*

1. 在resources目录下，创建applicationContext.xml，用于通过Spring自动注入需要的类

*<!-- 事务管理器 -->*

*<bean id="transactionManager" class="org.springframework.batch.support.transaction.ResourcelessTransactionManager"/>*

*<!-- 任务仓库 -->*

*<bean id="jobRepository" class="org.springframework.batch.core.repository.support.MapJobRepositoryFactoryBean">*

*<property name="transactionManager" ref="transactionManager"/>*

*</bean>*

*<!-- 任务加载器 -->*

*<bean id="jobLauncher" class="org.springframework.batch.core.launch.support.SimpleJobLauncher">*

*<property name="jobRepository" ref="jobRepository"/>*

*</bean>*

上面声明的transactionManager、jobRepository、jobLauncher，我们就可以执行批量任务啦！不过，我们还需要创建一个任务。在Spring Batch框架中，一个任务Job由一个或者多个步骤Step，而步骤又由读操作Reader、处理操作Processor、写操作Writer组成，下面我们分别创建它们。

1. 创建Reader

在resources目录下创建batch-data.csv文件，内容如下：

*1,PENDING*

*2,PENDING*

*3,PENDING*

*......*

从该文件中读文件。读操作需要实现ItemReader<T>接口，框架提供了现成的实现类FlatFileIt

emReader，使用该类需要设置Resource和LineMapper。Resource代表数据源，LineMapper表示如何将每行数据转成对应的DTO对象

* 创建Reader，命令数据使用DeviceCommand对象描述，代码如下：

*@Data*

*public class DeviceCommand {*

*private String id;*

*private String status;*

*}*

* 自定义LineMapper，将每行数据转换成DeviceCommand

*public class HelloLineMapper implements LineMapper<DeviceCommand> {*

*@Override*

*public DeviceCommand mapLine(String line, int lineNumber) throws Exception {*

*// 逗号分割每一行数据*

*String[] args = line.split(",");*

*// 创建DeviceCommand对象*

*DeviceCommand deviceCommand = new DeviceCommand();*

*// 设置id值到对象中*

*deviceCommand.setId(args[0]);*

*// 设置status值到对象中*

*deviceCommand.setStatus(args[1]);*

*// 返回对象*

*return deviceCommand;*

*}*

1. 创建Processor

读完数据后，通过ItemProcessor接口中的process进行处理

*public class HelloItemProcessor*

*implements ItemProcessor<DeviceCommand, DeviceCommand> {*

*@Override*

*public DeviceCommand process(DeviceCommand deviceCommand) throws Exception {*

*// 模拟下发命令给设备*

*System.out.println("send command to device, id=" + deviceCommand.getId());*

*// 更新命令状态*

*deviceCommand.setStatus("SENT");*

*// 返回命令对象*

*return deviceCommand；*

*}*

*}*

1. 创建Writer

处理完数据后，需要更新命令状态到文件中，与读文件类似需要实现ItemWriter<T>接口，框架中提供了FlatFileItemWriter实现类，在该类中设置Resource和LineAggregator，设置数据源及将DTO对象转成字符串保存到文件的每行

*public class HelloLineAggregator implements LineAggregator<DeviceCommand> {*

*@Override*

*public String aggregate(DeviceCommand deviceCommand) {*

*StringBuffer sb = new StringBuffer();*

*sb.append(deviceCommand.getId());*

*sb.append(",");*

*sb.append(deviceCommand.getStatus());*

*return sb.toString();*

*}*

*}*

1. 测试主程序

*public static void main(String[] args) throws Exception {*

*// 加载上下文*

*String[] configLocations = {"applicationContext.xml"};*

*ApplicationContext applicationContext = new ClassPathXmlApplicationContext(configLocations);*

*// 获取任务启动器*

*JobLauncher jobLauncher = applicationContext.getBean(JobLauncher.class);*

*JobRepository jobRepository = applicationContext.getBean(JobRepository.class);*

*PlatformTransactionManager transactionManager = applicationContext.getBean(PlatformTransactionManager.class);*

*// 创建reader*

*FlatFileItemReader<DeviceCommand> flatFileItemReader = new FlatFileItemReader<>();*

*flatFileItemReader.setResource(new FileSystemResource("src/main/resources/batch-data.csv"));*

*flatFileItemReader.setLineMapper(new HelloLineMapper());*

*// 创建processor*

*HelloItemProcessor helloItemProcessor = new HelloItemProcessor();*

*// 创建writer*

*FlatFileItemWriter<DeviceCommand> flatFileItemWriter = new FlatFileItemWriter<>();*

*flatFileItemWriter.setResource(new FileSystemResource("src/main/resources/batch-data.csv"));*

*flatFileItemWriter.setLineAggregator(new HelloLineAggregator());*

*// 创建Step*

*StepBuilderFactory stepBuilderFactory = new StepBuilderFactory(jobRepository, transactionManager);*

*Step step = stepBuilderFactory.get("step")*

*.<DeviceCommand, DeviceCommand>chunk(1)*

*.reader(flatFileItemReader)       // 读操作*

*.processor(helloItemProcessor)    // 处理操作*

*.writer(flatFileItemWriter)       // 写操作*

*.build();*

*// 创建Job*

*JobBuilderFactory jobBuilderFactory = new JobBuilderFactory(jobRepository);*

*Job job = jobBuilderFactory.get("job")*

*.start(step)*

*.build();*

*// 启动任务*

*jobLauncher.run(job, new JobParameters());*

*}*

执行后，在batch-data.csv文件中，写入内容：

*1,SENT*

*2,SENT*

*......*

至此实现了Spring Batch的简单实用。

https://www.ibm.com/developerworks/cn/java/j-lo-springbatch1/#artrelatedtopics

http://www.cnblogs.com/gulvzhe/archive/2011/10/25/2224249.html

https://www.ibm.com/developerworks/cn/java/j-lo-springbatch1/