# Spring Batch 4.1 译文

# 1.Spring 简介

很多企业级的应用都含有需要大批量数据处理的任务，这些业务场景包括：

1. 自动化的不需要用户干预的大批量信息处理任务，比如定时任务等；
2. 周期性的大批量数据重复处理；
3. 内部或者外部系统产生的大批量的需要数据以事务的方式格式化、校验、处理的任务等；

Spring Batch就是用来解决这些问题的框架，Spring Batch是基于Spring的基本框架的，所以可以使用Spring基本框架的一些特性，需要提前注意的是，Spring Batch并不是一个调度器框架，也不提供调度相关的功能，为了更好的解决上面的业务场景面临的问题，需要与一些调度框架一起使用，比如Quartz、Scheduler等。

## 1.1 框架开发背景

目前，开源软件组织与相关的团体主要关注Web与微服务架构领域的开源技术框架，而基于java的批处理框架并没有得到足够的关注与支持，传统的企业IT环境仍然面临着批处理的业务场景，由于缺乏一定的标准，每家公司研发的批处理框架架构可能完全基本不太一样，造成这一领域的标准缺失。

Spring开源组织与埃森哲公司都关注到这一点，决定结合2家的优势搞一个批处理任务处理的标准，所以Spring Batch的主要构想就是埃森哲的批处理架构技术+Spring框架特性。

## 1.2 使用场景

传统的批处理程序主要由3个部分组成：

* 读取大量数据，比如从数据库或者文件等；
* 处理数据；
* 写数据到某种媒介上；

Spring Batch框架完成了任务步骤的自动化处理，可以以一种集合的方式处理相似的事物

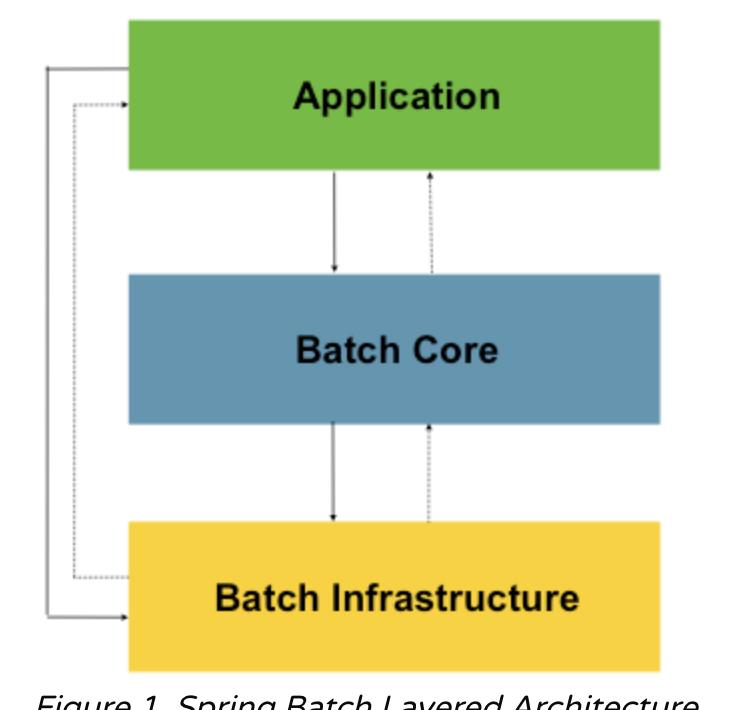
Spring Batch的提供的功能包括：

* 周期性运行批处理程序；
* 并发批处理运行；
* 消息批处理；
* 失败处理；
* 批处理程序的顺序处理与条件走向处理；
* 局部处理；
* 批处理事务；

技术目标

* 应用Spring的编程模型，让开发人员主要关注业务代码的开发，而不是基础环境；
* 应用程序分层，基础环境、批处理基础环境、应用逻辑代码；
* 面向接口编程，并提供了接口的简单实现；
* 配置简单，扩展性强，可定制化强；

## 1.3 Spring Batch体系结构



Application主要是开发人员写的代码，Batch Core主要包含运行批处理任务的相关的运行环境类，比如JobLaunch、Job、Step等，Batch基础环境包括Spring Batch提供的基础服务，可以由Spring框架使用的，应用代码中也可以使用。

## 1.4 批处理任务设计准则

批处理任务需要考虑的设计原则：

* 如果批处理架构与在线架构相互影响，尽量做到模块复用；
* 设计批处理任务时尽可能简单，避免有太复杂的业务逻辑；
* 批处理读取数据要尽可能快，也就是说，数据要离批处理任务尽可能的近；
* 近可能的减少使用I/O的次数，操作尽可能在内存中完成；
* 对I/O使用情况进行检查，避免不必要的I/O操作，通常来说，要避以下4类不必要的I/O操作：

1.当数据可以一次读取并缓存下来时避免每次事务都读一遍数据；

2.一个事务内，避免多次读取同样的数据；

3.不必要的表与索引读取；

4.SQL内没有where条件；

* 已有数据的结果不要计算2次，比如，报表数据，每次计算后，可以复用上一次的计算结果；
* 在批处理运行开始前，就分配足够运行的内存；
* 永远假设数据的准确性，加入必要的数据校验；
* 尽早进行压力测试；
* 定期备份；

## 1.5 批处理策略

开始设计一个批处理任务时，需要能够把他们分成一系列的步骤，步骤的主要的种类如下：

转换处理：就是将数据转化成规定的格式；

校验处理：校验处理保证输入输出的数据的正确性，校验信息传统上会写到文件头或者文件尾，里面可能包含的信息时校验结果总结，校验算法或者是记录级别的交叉校验；

提取处理：从输入中抽出满足条件的数据，或将满足条件的数据写出；

提取/更新处理：提取数据，根据提取的数据更新数据库的记录或者产生新的输出记录；

输出/格式化处理：格式化数据并产生输出数据；

另外，还要提供额外的业务逻辑操作命令，这些都需要额外开发；

Step可能包含的实用处理组件可能有：

Sort： 排序：

Split：分解，

Merge：合并；

批处理应用可以分为数据库批处理，文件批处理，消息队列批处理；

任何批处理系统的核心都是批处理策略，影响批处理策略的因素包括：

# 2.Spring Batch 4.1新增内容

Spring Batch 4.1发布版新增了以下的特性：

1.@SpringBatchTest注解用来简化batch组件的测试；

2.@EnableBatchIntegration注解来简化远程块分区配置；

3.新的JsonItemReader&JsonFileItemWriter来I/OJSON数据；

4.使用Bean Validation API校验数据；

5.支持JSR305注解规范；

6.FlatFileItemWriterBuilder类的增强。

详细内容略。

# 3. Batch组件

## 3.3 ExecutionContext

ExecutionContext就是键值对集合，用来存储Job/Step执行的上下文环境或者状态信息，它是StepExecution与JobExecution的一部分，不能独立存在。类似Quartz的JobDataMap，最好的使用的例子就是重启时记录到上次执行存下来的状态信息；比如当ItemReader读取文件时，定期存储读取的行的指针，当断电或者程序异常终止时，再次启动可以获得上一次记录到的读取的行指针；需要做的就是定期把行指针存到ExcutionContext中，

# 12 JSR-352支持

JSR-352是JSR组织发布的基于java平台的批处理规范。

## 12.1 Spring Batch与JSR-352规范之间的关系

Spring Batch与JSR-352规范定义基本是差不多，他们都有有steps组成的jobs，都有readers、processors、writers与listeners；但是他们的一些API是完全不同的，具体的例子可以看源代码。

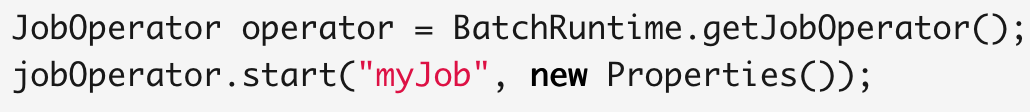
## 12.2 设置

### 12.2.1 应用程序上下文环境

所有的给予JSR-352的JOB都含有2个应用环境上下文，一个基础环境上下文，parent context，存储了Spring Batch环境的上下文信息，比如JobRepository等；一个Job上下文环境，类似于Spring Batch Job的JobExecutionContext里面内容；parent context里面的内容是通过框架内部的jsrBaseContext.xml来定义的，里面扽可容可以被JSR-352-BASE-CONTEXT系统属性覆盖。

### 12.2.2启动基于JSR-352规范的JOB

启动一个JSR-352规范的job是非常方长方便的，代码如下：



上面的代码虽然简单，但是因为上面的代码启动了一系列Spring Batch缺省配置的基础环境，很多都需要你进行定制化替代，BatchRuntime.getJobOperator()这行代码启动了一系列的Spring Batch基础配置，详细的就不列出了。

## 12.3 依赖注入

JSR-352规范基本是基于Spring Batch编程模型实现的，所以不需要进行任何额外的依赖注入配置，就可以使用Spring框架提供的依赖注入配置，Spring Batch支持由JSR-352定义的用于加载Batch组件的全部3个方法：

1.实现规范指定的JSR-352,Spring Batch是基于Spring的，所以支持在JSR-352规范内部进行依赖注入；

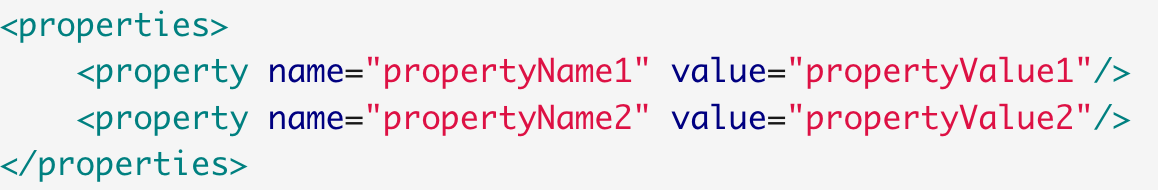
2.归档文件加载器，JSR-352规范定义batch.xml文件，用于提供逻辑名到一个真实class全限定路径的映射，这个文件在/META-INF目录下；

3.线程上下文类加载器，JSR-352规范通过配置类的全限定路径来配置批处理应用组件。

12.4 批处理属性

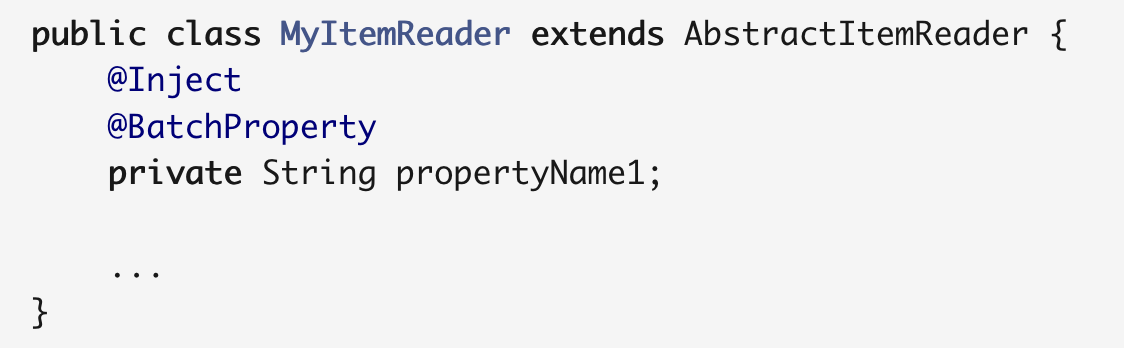
12.4.1 属性支持

JSR-352允许在JOB、Step与其他的batch组件内部定义属性，方式如下：



12.4.2 @BatchProperty 注解

上面的属性可以通过@Inject与@BatchProperty2个注解注入到其他的batch组件中，代码如下：



12.4.3 属性替换

可以通过简单的容器表达式来进行属性替换，普遍的用法是#{operator[‘key’]};

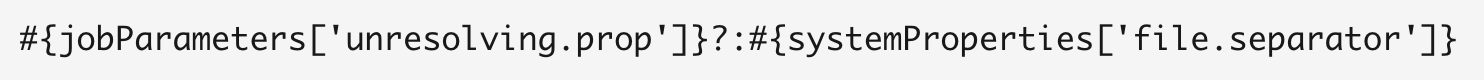
属性容器由：

jobParameters-

jobProperties-

systemProperties-

partitionPlan-



12.5 处理模型

JSR-352规范提供了与Spring Batch基本相同的2歌基本的处理模型：

1.用于做数据处理的Item；

2.用于做业务逻辑的Task，JSR规范的Task的接口是Batchlet，在Spring Batch下面是Tasklet；

12.5.1 Item

4.配置与运行一个Job4.1.2 Java配置

1.Batch语言定义

Spring Batch中使用的批处理概念与很多的成熟的批处理架构中的概念基本是差不多的，有Jobs、Steps、还有开发者自己开发的数据处理单元ItemReader与ItemWriter；但是由于Spring框架的操作习惯，Spring Batch与其他类型的批处理架构也有一些不同的地方，主要有以下的方面：

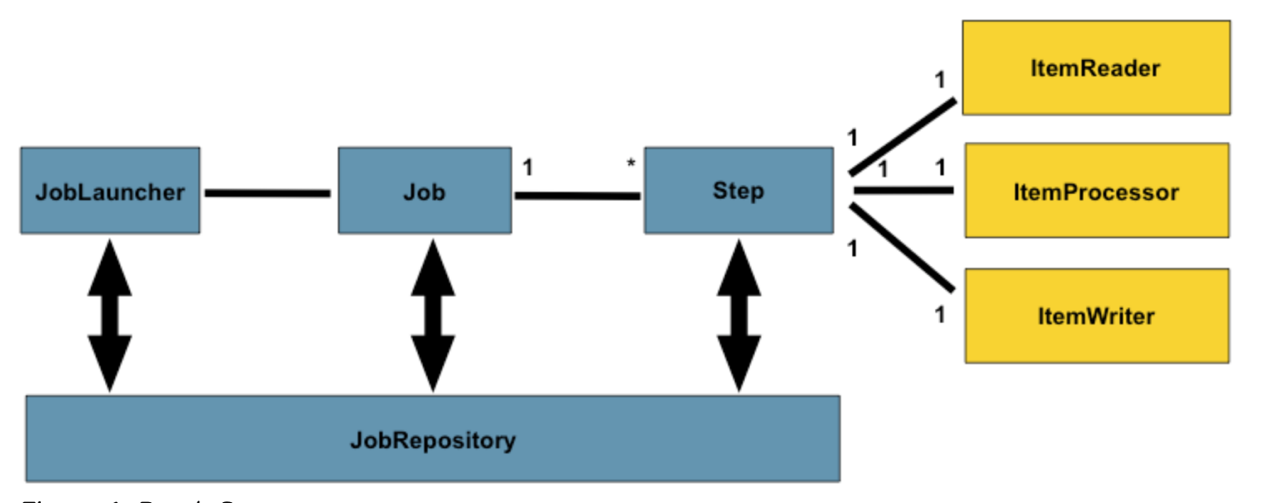
概念的分离；

分层架构；服务都是用了接口；

都提供了很多接口的简单的缺省的实现，可以快速的适配，使用简单；

扩展性很高；

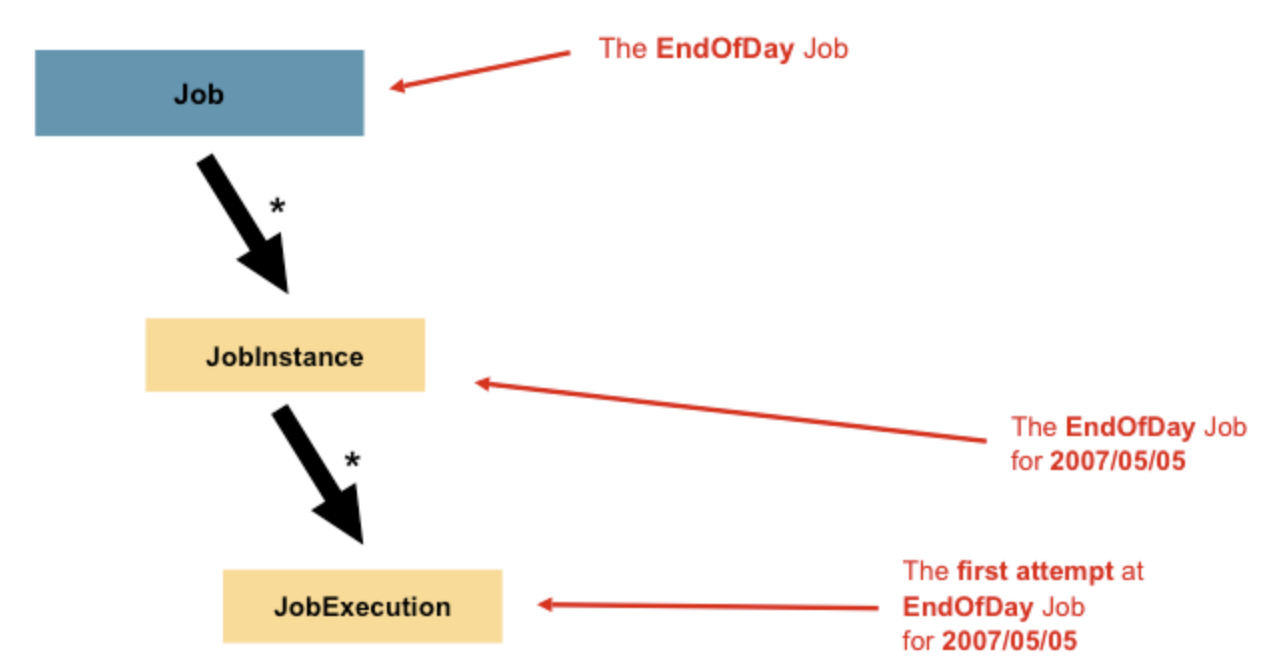
下面的图是使用了很多年的Batch处理程序架构图；



这个架构图里面包含了几乎所有的批处理过程中涉及到的组件；Spring框架根据这个蓝图，提供了其中所有组件的物理实现，这些实现都是健壮的，可维护性极高的组件系统，用来简化复杂的批处理应用。

1.1 Job

Job代表的就是完成一个批次处理。



在Spring Batch中，Job是Step实例的容器，他把很多的的Step组合成一个逻辑上的流，用来表达job中的处理步骤；可以在JOb中给所有的step添加一些全局配置属性；job的配置包括：

名字；

Step实例的定义与执行顺序；

job是否是可以重新启动的；

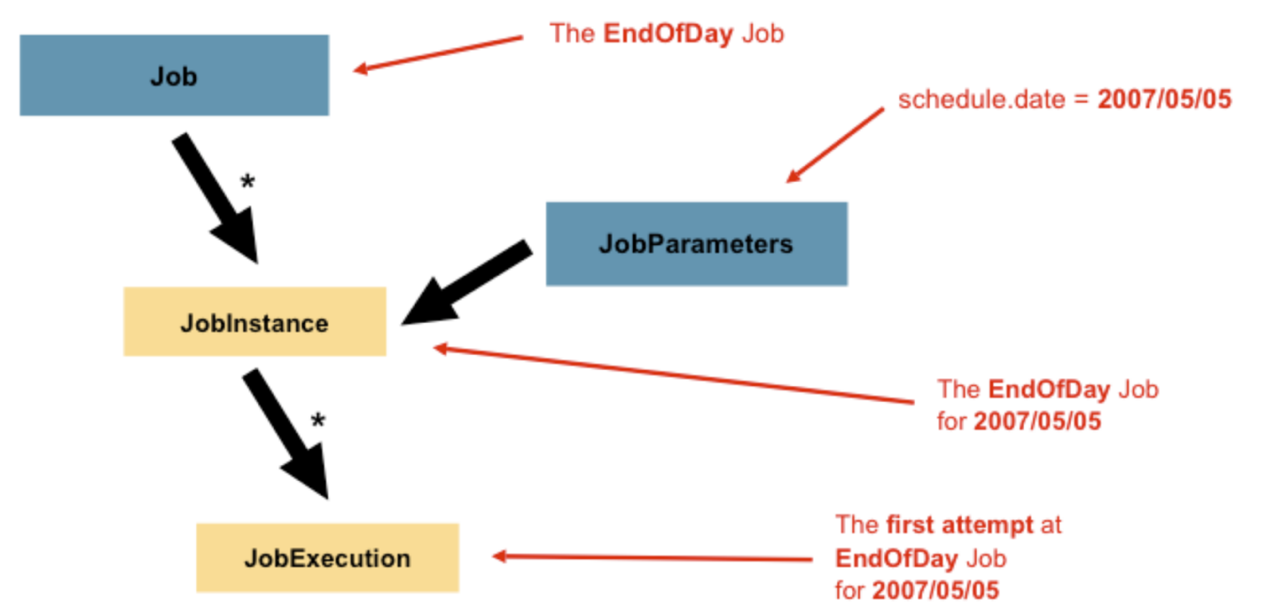
Spring提供了Job的简单实现SimpleJob类。

1.1.1 JobInstance

JobInstance的意思就是一次Job的运行，每一个JobInstance都可以多次执行；JobInstance的定义不关系数据的来源，完全由ItemReader来决定数据怎么读取；

1.1.2 JobParameters

JobParameters的作用如下图：



1.1.3 JobExecution

JobExecution就是指Job的一次运行，运行的结果可能是失败或者成功，但是关联这个JobExecution的JobInstance只有在JobExecution返回的结果是成功的时候才会被设置为complete的状态；JOb定义执行的任务是什么，JobInstance只是纯粹的组织执行情况的对象，主要是为了让Job可以方便的重启；JobExecution则代表了当任务执行时真正发生的事情，JObExecution包含了很多重要的属性；

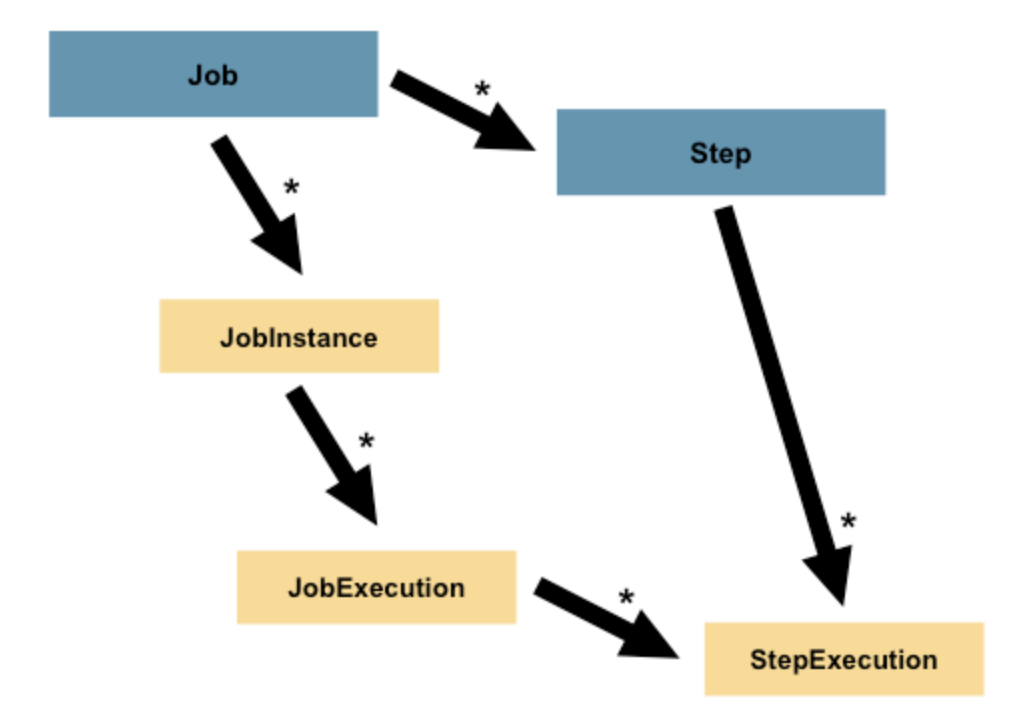
Status: 执行结果；

startTime: 开始执行的时间；

endTime: 执行结束的时间；

1.2 Step

Step时批次Job中一块任务，因此，每一个Job都是由一个或者多个的steps组成；Step包含所有的执行批处理任务的必要的信息；实际上，任何Step的内容都是由开发者控制的，开发者想怎么写Step就可以怎么写Step，Step也有自己的StepExecution，这个执行实例是与JobExecution相对应的。



Step表示对于Step的一次执行，每次Step启动的时候创建一个新的StepExecution，StepExecution中包含Step相关的，JobExecution相关的或者事务相关的数据，另外每个StepExecution还包括ExecutionContext对象，里面包括了开发者设置的任何的数据。

1.3 ExecutionContext

ExecutionContext里面是框架内部存储的key/value键值对的集合，主要是为了存储StepExecution或者JobExecution执行时的上下文的环境信息；有点类似于Quartz里面的JobDataMap；最好的使用案例就是让批处理应用重新执行；比如当使用文件作为batch的输入的时候，每次处理一行；

框架周期性的向ExecutionContext存储处理的行数，这样ItemReader就可以在错误发生时，读到输入的状态，你需要做的就是把读取的行数放到context中。



1.4 JobRepository

JobRepository为上面所提到的所有的模型提供了持久化存储机制，

1.3 控制Step流

job拥有组织steps的能力也就拥有

6.1 ItemReaders与ItemWriters

所有的批处理程序的计算形式都是读取大量数据->计算或者做数据转换->写出结果；Spring Batch提供了3个主要的接口用于批量数据的读、处理、与写：ItemReader、ItemProcessor、ItemWriter；

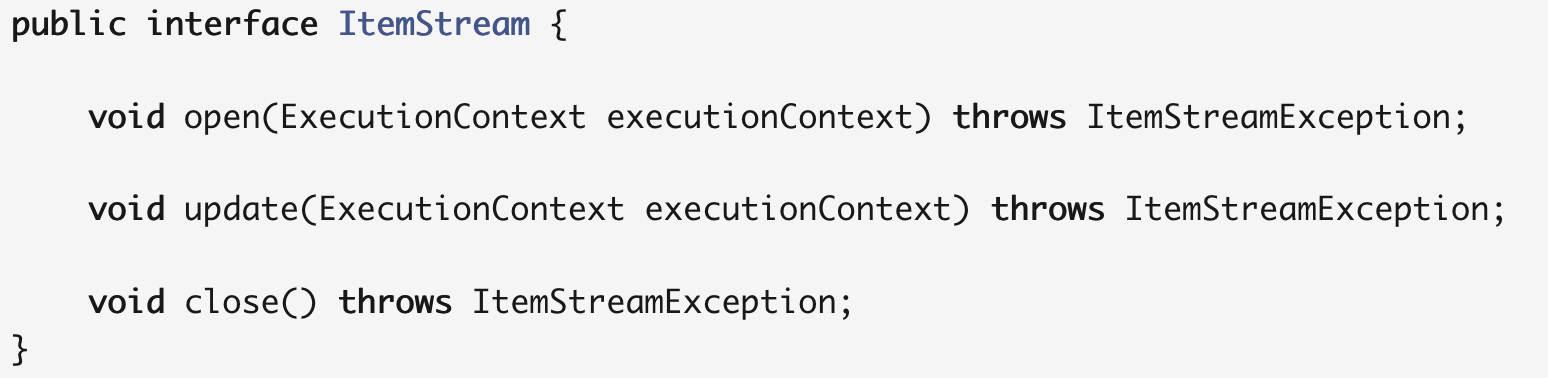
6.1.3.2 过滤记录

ItemProcesser的传统的使用就是过滤传递给ItemWriter的记录；过滤与忽略时明显不同的，忽略指的是记录是无效的，忽略掉，而过滤是指记录是有效的，但是不能被传递给Wiriter用于输出；

6.1.3.3 Fault Tolerance（容错性）

6.1.3.4 ItemStream

ItemReader与ItemWriter的单一职责设计的非常高效，但是单一的职责使得他们局限在自己的操作上，对于使用这些接口的一些相同的操作，就需要另外的接口来实现；通常来说，作为job生命周期内的一部分，readers&writers需要被打开、关闭、或者需要一些机制来持久存储状态；ItemStream接口就是用于这个目的的。



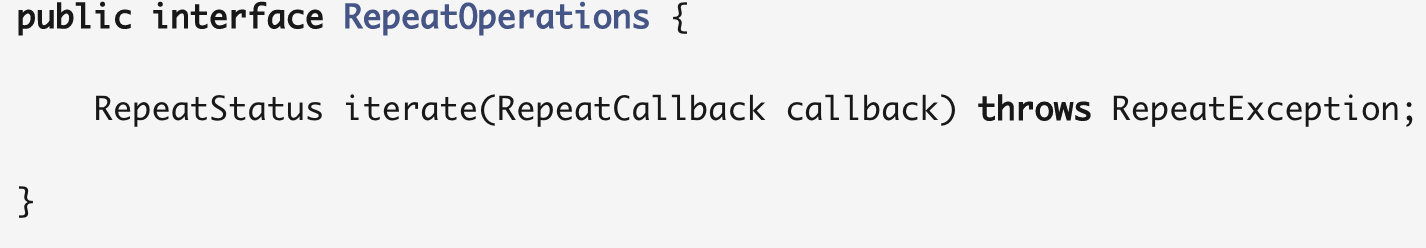
ExecutionContext这个执行环境上下文，ItemReader的实现类，如果也实现了ItemStream接口，都应该在调用read前调用open()方法；这是为了打开资源或者是获得一个到数据库的连接，如果ItemWriter的实现类也实现了ItemStream的时候也是如此；

6.1.10 Database

8 repeat

8.1.1 RepeatTemplate

批处理都是可以重复进行的行为，不仅是从优化处理的觉度，或者是job本身来说，为了提供重复进行的功能，Spring Batch提供了RepeatOperations接口，接口的定义如下：

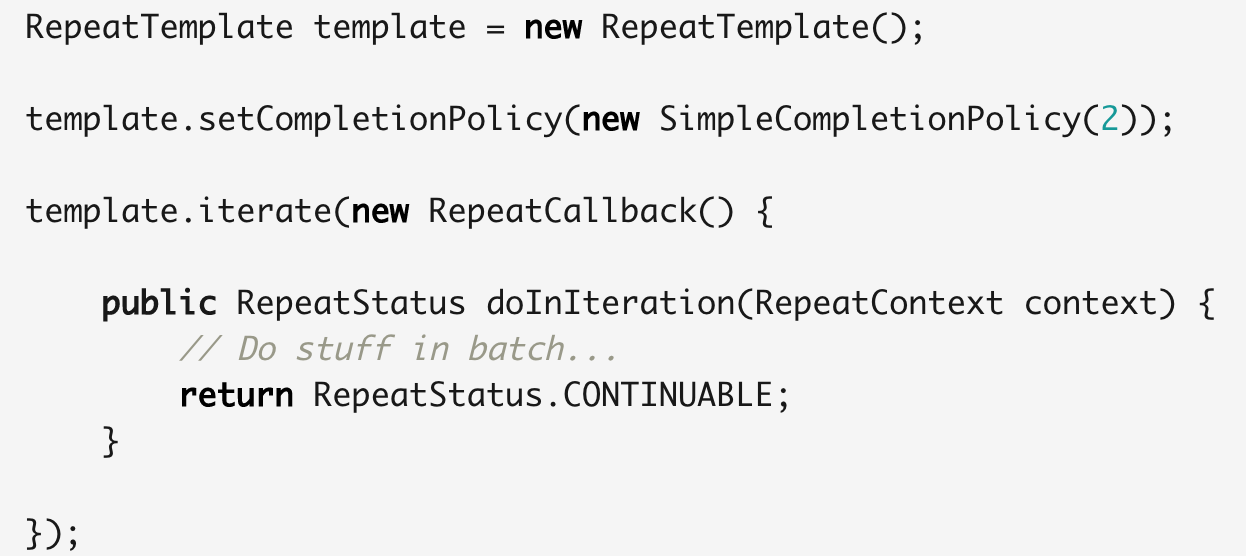


回调接口如下：



可以让你重复执行一些业务逻辑。回调回重复执行，直到实现确定迭代应该终止时，返回的值是一个枚举类型的RepeatStatus数据，RepeatStatus返回给状态给调用者，用于告知是否还有更多的工作要做，通常来说，RepeatOperations的实现因该检查RepeatStatus的状态，并把它作为是否终止迭代的条件之一；如果回调要告诉调用者没有更多的工作要做的话，就返回RepeatStatus.FINISHED。

RepeatOperations的最简单的实现类就是RepeatTemplate，例子如下：



8.1.1.1 RepeatContext

RepeatCallback回调接口的方法的参数时RepeatContext，很多回调都会忽略这个context，如果有必要，它可以作为一个迭代过程中的属性包，当iterate方法结束时，RepeatContext也就生命周期结束了。

8.1.1.2 RepeatStatus

8.1.2 完成策略

在一个RepeatTemplate内部，iterate方法循环的终止是由ComletionPolicy决定的，ComletionPolicy负责产生并更新RepeatContext，每次迭代结束后，RepeatTemplate询问ComletionPolicy是否迭代要终止；其实现是SimpleCompletionPolicy；

8.1.3 异常处理

9.1 retry

为了让处理更健壮并产生较少的失败，有时候重新执行失败的任务接下来就可能会成功，造成失败的问题也许是偶尔发生的，比如对web service的远程调用，可以由于当时网络的堵塞或者是因为数据库的DeadlockLoserDataAccessException而造成失败。

9.1.1 RetryTemplate

12.1 JSR-352支持

因为Spring Batch3.0版本以及之后的版本已经完全实现了对JSR-352的支持。

12.1.1 关于Spring Batch与JSR-352关系的通用说明

Spring Batch与JSR-352在结构上是相同的，他们都有jobs\steps\readers\processors\writers\listeners\等，但是也有一些不同的地方，比如根据Spring Batch接口开发的实现工作在JSR规范的job中时，以JSR规范的JOB形式为主，以JSR规范的接口开发的实现不能工作在Spring Batch的job中。

12.2 创建

12.2.1 应用上下文环境

在Spring Batch中的所有的基于JSR-352规范开发的JOB都包含2个上下文环境，parent Context，主要包含了Spring Batch的基础的环境信息，比如JobRepository等，child Context包含运行的job的配置信息。

12.2.2 在Spring Batch中启动JSR-352规范的JOB

启动的代码如下：



12.3 依赖注入

JSR-352规范模型也是基于Spring batch的编程模型，在这种情况下，虽然没有明确的指明需要一个正式的DI实现，但是DI依然是在后面实现了的，Spring Batch支持用于加载batch配置文件的全部的3个方法。

7 Spring Batch的并行处理

很多Spring Batch的处理难题都可以通过使用但线程，单job的方式解决，如果我们现在要考虑一个更复杂的实现之前，就要考虑单线程的方式是否已经满足现在的要求，首先，要测量在最简单情况下的job的运行性能，如果已经满足给定的性能要求，则没有必要使用更复杂的多线程的技术。

当准备开发一个并行处理的job时，Spring Batch提供了很多的选项，这些选项将在本小结讨论，有2种并行处理的模型：

1.一次处理，多个线程

2.多次处理；

在Spring Batch中可以简单的分为如下几种情况：

1.Step多线程处理（一次处理）；

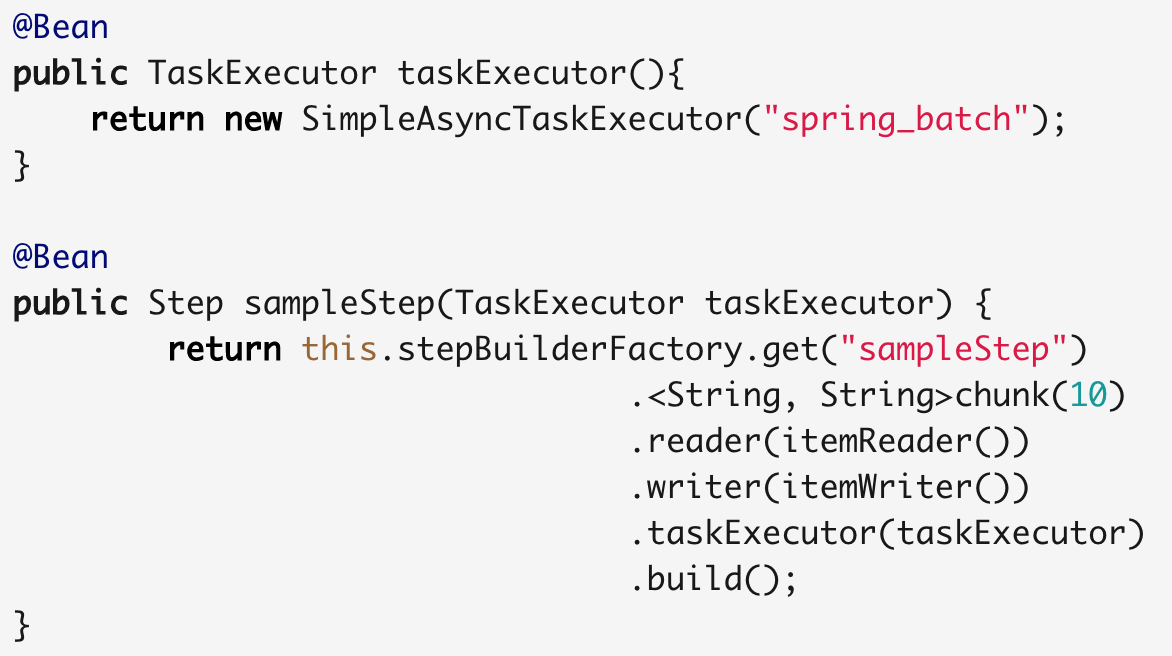
2.并行Step(一次处理)；

3.远程Step批量处理（多次处理）；

4.Step分区（一次或者多次处理）。

7.1 多线程Step

启动多线程Step的最简单的方式就是在配置Step的时候传入TaskExecutor对象，典型的例子如下：



在上面的配置中呢把每个chunk的item数放到一个线程中去执行，这意味着item的处理顺序是不固定的，还可以在配置中限制线程的数量。

例子如下：



但是使用多线程的Step也有一些实际的限制，Step中的很多内容都是存储状态的，比如Reader与Writer；如果状态被线程隔离了，这些组件在多线程的Step中可能就没法使用了，实际来说，Spring Batch提供的大部分的readers&writers都是面向多线程的，但是也有无状态的readers&writers或者是线程安全的readers&writers。

7.2 并行的Steps

可以并行化的应用逻辑可以分开放入到单独的Step中，这些Step可以在JOB中并行执行，Step的并行是非常容易配置的，如下图：

