# Spring-batch

## Hello World

环境：

Jdk：1.6

Maven：3.2.5

Jar-version：

|  |
| --- |
| <spring.batch.version>2.2.1.RELEASE</spring.batch.version>  <spring.version>3.2.4.RELEASE</spring.version> |

### 工程配置文件

Spring-batch基础信息配置：

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xmlns:p=*"http://www.springframework.org/schema/p"*  xmlns:tx=*"http://www.springframework.org/schema/tx"*  xmlns:aop=*"http://www.springframework.org/schema/aop"*  xmlns:context=*"http://www.springframework.org/schema/context"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans*  *http://www.springframework.org/schema/beans/spring-beans-3.0.xsd*  *http://www.springframework.org/schema/tx*  *http://www.springframework.org/schema/tx/spring-tx-3.0.xsd*  *http://www.springframework.org/schema/aop*  *http://www.springframework.org/schema/aop/spring-aop-3.0.xsd*  *http://www.springframework.org/schema/context*  *http://www.springframework.org/schema/context/spring-context-2.5.xsd"*  default-autowire=*"byName"*>    <bean id=*"jobRepository"*  class=*"org.springframework.batch.core.repository.support.MapJobRepositoryFactoryBean"*>  </bean>  <bean id=*"jobLauncher"*  class=*"org.springframework.batch.core.launch.support.SimpleJobLauncher"*>  <property name=*"jobRepository"* ref=*"jobRepository"*/>  </bean>  <bean id=*"transactionManager"*  class=*"org.springframework.batch.support.transaction.ResourcelessTransactionManager"*/>  </beans> |

作业配置文件：

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <bean:beans xmlns=*"http://www.springframework.org/schema/batch"*  xmlns:bean=*"http://www.springframework.org/schema/beans"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xmlns:p=*"http://www.springframework.org/schema/p"*  xmlns:tx=*"http://www.springframework.org/schema/tx"*  xmlns:aop=*"http://www.springframework.org/schema/aop"*  xmlns:context=*"http://www.springframework.org/schema/context"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans*  *http://www.springframework.org/schema/beans/spring-beans-3.0.xsd*  *http://www.springframework.org/schema/tx*  *http://www.springframework.org/schema/tx/spring-tx-3.0.xsd*  *http://www.springframework.org/schema/aop*  *http://www.springframework.org/schema/aop/spring-aop-3.0.xsd*  *http://www.springframework.org/schema/context*  *http://www.springframework.org/schema/context/spring-context-2.5.xsd*  *http://www.springframework.org/schema/batch*  *http://www.springframework.org/schema/batch/spring-batch-2.2.xsd"*>  <bean:import resource=*"classpath:example02/job-context.xml"*/>  <job id=*"resourceJob"*>  <step id=*"resourceStep"*>  <tasklet transaction-manager=*"transactionManager"*>  <chunk reader=*"csvItemReader"* writer=*"csvItemWriter"*  processor=*"resourceProccessor"* commit-interval=*"2"*>  </chunk>  </tasklet>  </step>  </job>  <!-- 读取信用卡账单文件,CSV格式 -->  <bean:bean id=*"csvItemReader"*  class=*"org.springframework.batch.item.file.FlatFileItemReader"*  scope=*"step"*>  <bean:property name=*"resource"*  value=*"file:D:/data/batch/example02/widget.csv"*/>  <bean:property name=*"lineMapper"*>  <bean:bean  class=*"org.springframework.batch.item.file.mapping.DefaultLineMapper"*>  <bean:property name=*"lineTokenizer"* ref=*"lineTokenizer"*/>  <bean:property name=*"fieldSetMapper"*>  <bean:bean class=*"org.springframework.batch.item.file.mapping.BeanWrapperFieldSetMapper"*>  <bean:property name=*"prototypeBeanName"* value=*"widgetResource"*>  </bean:property>  </bean:bean>  </bean:property>  </bean:bean>  </bean:property>  </bean:bean>  <!-- lineTokenizer -->  <bean:bean id=*"lineTokenizer"*  class=*"org.springframework.batch.item.file.transform.DelimitedLineTokenizer"*>  <bean:property name=*"delimiter"* value=*","*/>  <bean:property name=*"names"*>  <bean:list>  <bean:value>id</bean:value>  <bean:value>docTitle</bean:value>  <bean:value>productCode</bean:value>  <bean:value>appCode</bean:value>  <bean:value>resourceId</bean:value>  </bean:list>  </bean:property>  </bean:bean>    <!-- 写信用卡账单文件，CSV格式 -->  <bean:bean id=*"csvItemWriter"*  class=*"org.springframework.batch.item.file.FlatFileItemWriter"*  scope=*"step"*>  <bean:property name=*"resource"* value=*"file:D:/data/batch/example02/outputFile.csv"*/>  <bean:property name=*"encoding"* value=*"GBK"*/>  <bean:property name=*"lineAggregator"*>  <bean:bean  class=*"org.springframework.batch.item.file.transform.DelimitedLineAggregator"*>  <bean:property name=*"delimiter"* value=*","*></bean:property>  <bean:property name=*"fieldExtractor"*>  <bean:bean  class=*"org.springframework.batch.item.file.transform.BeanWrapperFieldExtractor"*>  <bean:property name=*"names"*  value=*"id,docTitle,productCode,appCode,resourceId"*>  </bean:property>  </bean:bean>  </bean:property>  </bean:bean>  </bean:property>  </bean:bean>    <bean:bean id=*"widgetResource"* scope=*"prototype"*  class=*"com.example.batch.example02.WidgetResource"*>  </bean:bean>  <bean:bean id=*"resourceProccessor"* scope=*"step"*  class=*"com.example.batch.example02.ResourceProccessor"*>  </bean:bean>  </bean:beans> |

### Java对象

|  |
| --- |
| **package** com.example.batch.example02;  **public** **class** WidgetResource {  **private** String id;    **private** String docTitle;    **private** String productCode;    **private** String appCode;    **private** String resourceId;  **public** String getId() {  **return** id;  }  **public** **void** setId(String id) {  **this**.id = id;  }  **public** String getDocTitle() {  **return** docTitle;  }  **public** **void** setDocTitle(String docTitle) {  **this**.docTitle = docTitle;  }  **public** String getProductCode() {  **return** productCode;  }  **public** **void** setProductCode(String productCode) {  **this**.productCode = productCode;  }  **public** String getAppCode() {  **return** appCode;  }  **public** **void** setAppCode(String appCode) {  **this**.appCode = appCode;  }  **public** String getResourceId() {  **return** resourceId;  }  **public** **void** setResourceId(String resourceId) {  **this**.resourceId = resourceId;  }  } |

### 文件处理Processer

|  |
| --- |
| **package** com.example.batch.example02;  **import** org.springframework.batch.item.ItemProcessor;  **public** **class** ResourceProccessor **implements** ItemProcessor<WidgetResource, WidgetResource>{  @Override  **public** WidgetResource process(WidgetResource item) **throws** Exception {  item.setAppCode(item.getAppCode());  item.setDocTitle(item.getDocTitle()+"==");  item.setId(item.getId());  item.setProductCode(item.getProductCode());  item.setResourceId(item.getResourceId());  **return** item;  }  } |

### Test

|  |
| --- |
| package com.example.batch.ex02;  import org.springframework.batch.core.Job;  import org.springframework.batch.core.JobExecution;  import org.springframework.batch.core.JobParameters;  import org.springframework.batch.core.launch.JobLauncher;  import org.springframework.context.ApplicationContext;  import org.springframework.context.support.ClassPathXmlApplicationContext;  public class JobLaunch {  public static void main(String[] args) {  ApplicationContext context = new ClassPathXmlApplicationContext("example02/job/job.xml");  JobLauncher launcher = (JobLauncher) context.getBean("jobLauncher");  Job job = (Job) context.getBean("resourceJob");  try {  JobExecution result = launcher.run(job, new JobParameters());  System.out.println(result.toString());  } catch (Exception e) {  e.printStackTrace();  }  }  } |

### 概念说明

Job Repository：作业仓库，负责job、step执行过程的状态保存

Job Launcher：作业调度器，提供执行job的入口

Job：作业，由多个step组成，封装整个批处理操作

Step：作业执行的一个环节

Tasklet：Step具体执行逻辑操作，可以重复执行，可以设置具体的同步，异步操作。

Chunk:给定数量Item的集合，可以定义对chunk的读、处理、写操作，提交间隔等。

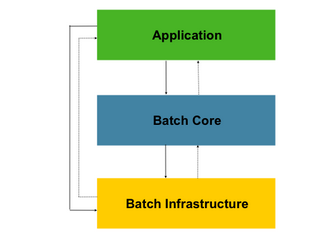
Item：一条记录数据

ItemReader：从数据源（文件系统、数据库、队列等）中读取Item

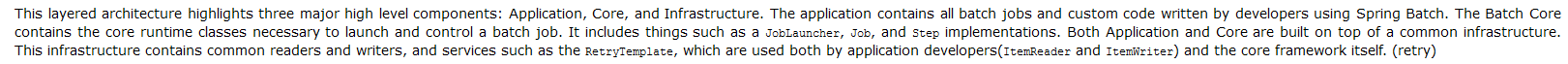
ItemProcesser：在Item写入数据源前，对数据进行处理

ItemWriter：将Item批量写入数据源

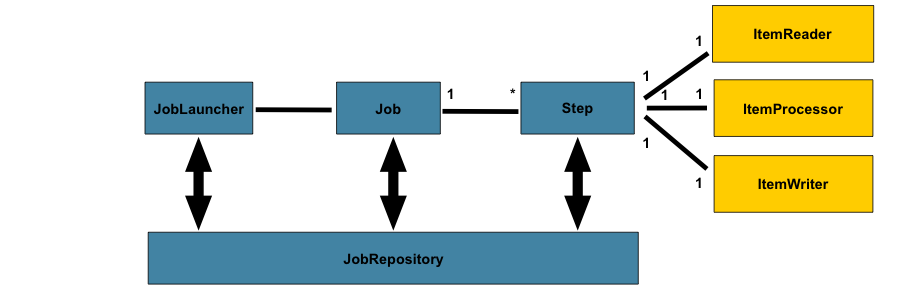
### Batch层级结构



官网释义：



## Spring-batch基本概念



Spring-batch架构图

一个job可以有一个或者多个step，每个step对应一个ItemReader、ItemProcessor、ItemWriter。

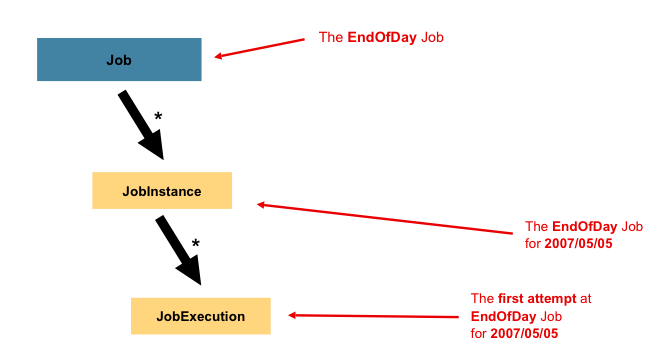
jobLauncher用来启动job，启动job需要从jobRepository获取存在的JobExecution(当前运行的job及step的结果和状态都保存在jobRepository中)。

### Job

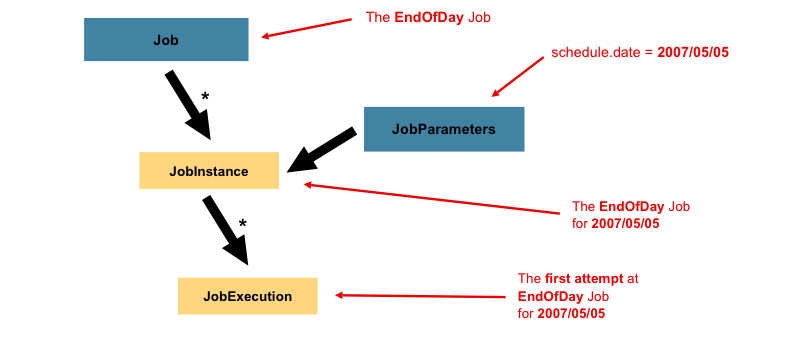
在spring-batch框架中，job是step的一个简单的容器，可以包含多个step，并且可以定义step的执行顺序，是否可以重启等。

### Job Instance

Job Instance是Job运行时的一个逻辑概念。每运行一个Job都会产生一个Job Instance。



### Job Parameters



如何区分job Instance，答案就是job parameters。如上图，有两job Instance，所以可以粗暴的认识job instance = job +job parameters。

### Job Execution

Job Execution表示job执行的句柄。一次job执行，可能成功也可能失败，除非execution 成功完成，相应的job instance才能完成。

Job，定义任务和执行步骤

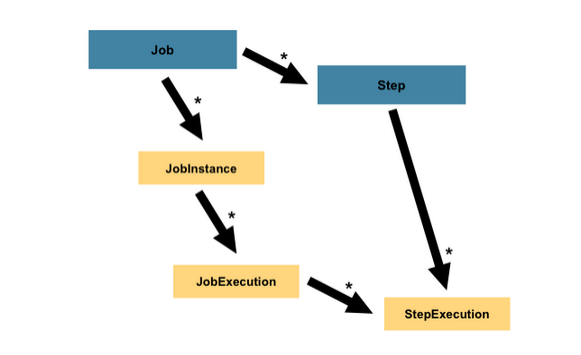
Job instance 组织任务执行，并确保任务能够正确重启

Job execution 存储运行时信息的机制。



### Step

Step是Job中独立有持续的一个步骤。一个Job可以包含一个或者多个step。Step包含一个batch process中所以必须的定义、控制信息。



### StepExecution

StepExecution 是step执行句柄。和jobExecution相似，stepExecution 在每次step运行时被创建。如果step执行失败的原因是上一步step执行失败，那么stepExecution将不会被持久化。stepExecution只有在step真正执行的时候才被创建。



### ExecutionContext

ExecutionContext是框架提供的持久化与控制的key/value对，能够让开发者在stepExecution 或jobExecution中保存需要进行持久化的状态。最好的用处就是帮助任务重启。用处理text文本文件举个例子，每处理一行，框架会周期性的在任务提交点(commit points)持久化ExecutionContext。这可以让ItemReader在运行期间遇到致命错误或者异常时存储它的状态，或者强制退出任务执行。

这需要把当前读取的行数放入上下文中，框架会将状态传递。

|  |
| --- |
| executionContext.putLong(getKey(LINES\_READ\_COUNT), reader.getPosition()); |

### JobRepository

JobRepository是spring-batch框架提供的用来持久化job执行期的元数据（元数据指job Instance、Job Execution、Job Parameters、Step Execution、Execution Context等数据），并提供两种默认实现。

### JobLauncher

JobLauncher 是一个通过给定JobParameters运行Job的一个简单接口，能够从JobRepository获取有效的JobExecution，并执行job。

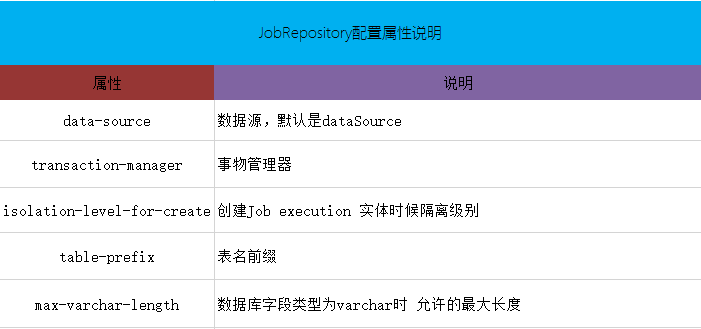
### Batch Namespace

|  |
| --- |
| <beans:beans xmlns="http://www.springframework.org/schema/batch"  xmlns:beans="http://www.springframework.org/schema/beans"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="  http://www.springframework.org/schema/beans  http://www.springframework.org/schema/beans/spring-beans.xsd  http://www.springframework.org/schema/batch  http://www.springframework.org/schema/batch/spring-batch-2.2.xsd">  <job id="ioSampleJob">  <step id="step1">  <tasklet>  <chunk reader="itemReader" writer="itemWriter" commit-interval="2"/>  </tasklet>  </step>  </job>  </beans:beans> |

## 配置作业

### JobRepository

|  |
| --- |
| <job-repository id=*"jobRepository"* data-source=*"dataSource"*  transaction-manager=*"transactionManager"* isolation-level-for-create=*"SERIALIZABLE"*  table-prefix=*"BATCH\_"* max-varchar-length=*"1000"* /> |



为了保证job在失败后重启所需的元数据包括必要的状态能够正确的持久化，需要配置事物管理器，也可是使用AOP配置存储行为的事物。

|  |
| --- |
| <!-- 事务管理器 -->  <beans:bean id=*"transactionManager"* class=*"org.springframework.jdbc.datasource.DataSourceTransactionManager"*>  <beans:property name=*"dataSource"* ref=*"dataSource"*></beans:property>  </beans:bean>    <aop:config>  <aop:advisor advice-ref=*"txAdvicer"* pointcut=*"execution(\* org.springframework.batch.core..\*Repository+.\*(..))"*/>  </aop:config>    <tx:advice id=*"txAdvicer"* transaction-manager=*"transactionManager"*>  <tx:attributes>  <tx:method name=*"\*"*/>  </tx:attributes>  </tx:advice> |

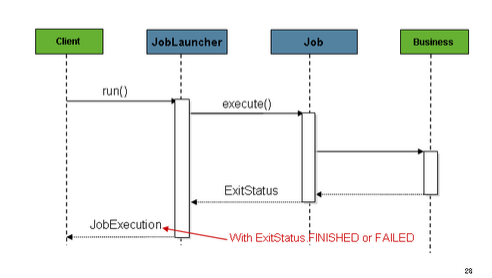
Spring-batch也支持不将这些领域对象持久化的配置：

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

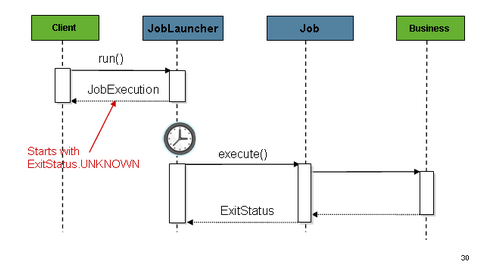
这个配置会将数据存储在内存当中，这种方式是不稳定的，而且在JVM 实例间是不支持重启的。

### JobLauncher

|  |
| --- |
| <beans:bean id=*"jobLaucher"* class=*"org.springframework.batch.core.launch.support.SimpleJobLauncher"*>  <beans:property name=*"jobRepository"* ref=*"jobRepository"*></beans:property>  </beans:bean> |



我们可以通过执行job，最终获取jobExecution，如上图，这是同步的方式，joblauncher还支持异步调用模式。



在实际的job作业中，可能由于batch处理时间过长，不可能让客户端一直处于等待状态，所以可以通过异步的方式调用作业。配置如下：

|  |
| --- |
| <beans:bean id=*"jobLaucher"* class=*"org.springframework.batch.core.launch.support.SimpleJobLauncher"*>  <beans:property name=*"jobRepository"* ref=*"jobRepository"*></beans:property>  <beans:property name=*"taskExecutor"*>  <beans:bean class=*"org.springframework.core.task.SimpleAsyncTaskExecutor"*></beans:bean>  </beans:property>  </beans:bean> |