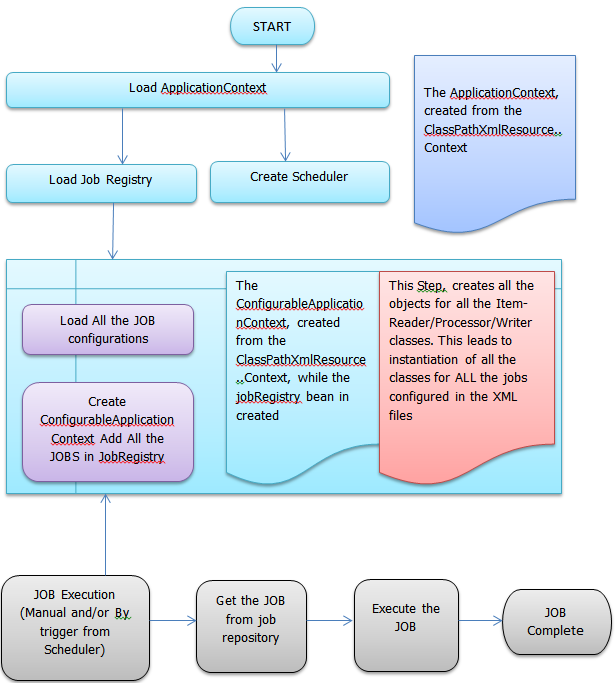
**Batch Configuration for Prototype Scoped Spring beans (Reader, Processor, and Writer)**

The execution of a STEP in batch framework is not controlled by the framework; rather, it is controlled by the ItemReader object. The framework keeps on restarting the steps once the writer is finished. To control it, the ETL jobs are keeping a Boolean variable to check, whether the reader has been executed once.

To achieve this, we need to have new object of Item Reader class every time the JOB is running.

**Flow of Framework:**

While start-up, the ApplicationContext is created from the Spring framework. Once the application context is getting created, it loads the configuration of “jobRegistry”. This job registry loads all the xml configuration files provided on the value list of “jobPaths”.



The above configuration leads to a problem of re-instantiation of all the classes configured as Item-reader/processor/writer beans. The “jobRegistry” always reloads the xml files and creates new instance of all the beans. Also, the garbage-collection does not happen as the same frequency, the objects are getting created.

**Configuration in current implementation (using batch version 2.0.4):**

Job Registry:

<bean id="jobRegistry" class="org.springframework.batch.core.configuration.support.ClassPathXmlJobRegistry">

<property name="jobPaths">

<list>

<value>/geneva/jobs-geneva.xml</value>

</list>

</property>

</bean>

Job Repository:

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

<property name="dataSource" ref="genevaDataSource" />

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

<property name="incrementerFactory">

<bean class="com.mercuria.etl.platform.CustomIncrementerFactory">

<property name="dataSource" ref="genevaDataSource" />

</bean>

</property>

</bean>

Parent Beans for JOB and Step:

<bean id="simpleJob" class="org.springframework.batch.core.job.SimpleJob"

abstract="true">

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

<property name="restartable" value="true" />

</bean>

<bean id="simpleStep"

class="org.springframework.batch.core.step.item.SimpleStepFactoryBean"

abstract="true">

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

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

<property name="startLimit" value="100" />

<property name="commitInterval" value="1" />

</bean>

<bean id="jobDetail\_PWR\_Balancing\_ReserveUtil" class="org.springframework.scheduling.quartz.JobDetailBean">

<property name="jobClass" value="com.mercuria.etl.platform.JobLauncherDetails" />

<property name="group" value="quartz-batch" />

<property name="jobDataAsMap">

<map>

<entry key="jobName" value="job\_PWR\_Balancing\_ReserveUtil" /> <entry key="jobLocator" value-ref="jobRegistry" />

<entry key="jobLauncher" value-ref="jobLauncher" />

</map>

</property>

</bean>

<bean id="job\_PWR\_IGCC\_Settlement\_Price" parent="simpleJob">

<property name="steps">

<list>

<!-- IGCC-Settlementprice -->

<bean parent="simpleStep">

<property name="itemReader">

<bean class="com.mercuria.etl.gva.tara.ReserveUtilizationReader">

...

</bean>

</property>

<property name="itemProcessor">

<bean class="com.mercuria.etl.gva.tara.BalancingPricesProcessor" />

</property>

<property name="itemWriter" >

<bean class="com.mercuria.etl.gva.tara.IGCCSettlementPriceWriter">

...

</bean>

</property>

</bean>

</list>

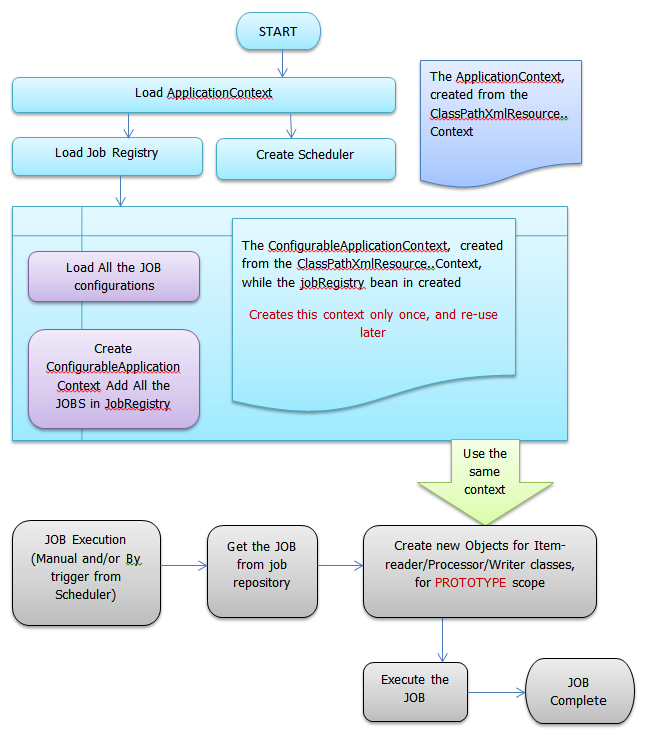
</property>

</bean>

Typically, most of the JOBs are using RecordSetWriter object as a writer. As the complete xml is loaded at the start of every JOB, there are huge number of instances are getting created in JVM. We are expecting that, this is leading to a Memory-Leak issue in ETL JOB. For this reason, sometimes jobs are getting stuck.

**New Configuration (Batch version 2.1.9):**

To stop reloading the complete JOB configurations everytime the JOB get executed, we can use “singleton” scope for JOB and Steps configurations. We can use “prototype” scope of the Item-reader/processor/writer beans. At the start of JobRegistry, we can load the job xml configurations only once. While any job is in execution, it only creates objects for its own configurations, instead of loading the complete configuration again and again. This shall reduce the number of objects live in JVM.



**Sample Configuration:**

Say, for example:

A, D – are ItemReader classes

B – is ItemProcessor

C – is ItemWriter

We keep the same configurations for “jobRepository”, “simpleJob” and “simpleStep”, but we change the “jobRegistry” from “ClassPathXmlJobRegistry” to “MapJobRegistry”. We add a bean of “*AutomaticJobRegistrar*” to load the job configuration XML only at the start-up of AoolicationContext. We mark all the JOB bean as PROTOTYPE scope. We call a “System.gc()” at the time we register each JOB.

<bean id=*"jobRegistry"* lazy-init=*"false"*

class=*"org.springframework.batch.core.configuration.support.MapJobRegistry"* />

<bean

class=*"org.springframework.batch.core.configuration.support.AutomaticJobRegistrar"*>

<property name=*"applicationContextFactories"*>

<bean

class=*"org.springframework.batch.core.configuration.support.ClasspathXmlApplicationContextsFactoryBean"*>

<property name=*"resources"* value=*"classpath\*:/bpp-context.xml"* />

</bean>

</property>

<property name=*"jobLoader"*>

<bean

class=*"test.profile.batch.PrototypeJobLoader"*>

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

</bean>

</property>

</bean>

We add a factory bean “JobStepFactoryBean” for each step bean, which are PROTOTYPE scope.

<bean id=*"step\_Test\_Hello\_01"* parent=*"simpleStep"* scope=*"prototype"* >

<property name=*"itemReader"*>

<bean class=*"test.profile.A"* />

</property>

<property name=*"itemProcessor"*>

<bean class=*"test.profile.B"*></bean>

</property>

<property name=*"itemWriter"*>

<bean class=*"test.profile.C"*></bean>

</property>

</bean>

<bean id=*"job\_Test\_Hello\_01"* parent=*"simpleJob"* scope=*"prototype"*>

<property name=*"steps"*>

<list>

<bean class=*"test.profile.batch.JobStepFactoryBean"*>

<constructor-arg>

<value>step\_Test\_Hello\_01</value>

</constructor-arg>

</bean>

</list>

</property>

</bean>

This creates only those objects are required to execute the JOB.