使用 Spring Batch

MyBatis-Spring 1.1.0 发布以后，提供了三个 bean 以供构建 Spring Batch 应用程序：`MyBatisPagingItemReader`、`MyBatisCursorItemReader` 和 `MyBatisBatchItemWriter`。而在 2.0.0 中，还提供了三个建造器（builder）类来对 Java 配置提供支持：`MyBatisPagingItemReaderBuilder`、`MyBatisCursorItemReaderBuilder` 和 `MyBatisBatchItemWriterBuilder`。

提示 本章是关于 Spring Batch 的，而不是关于 MyBatis 的批量 `SqlSession`。要查找关于批量 session 的更多信息，请参考 使用 SqlSession 一章。

MyBatisPagingItemReader

这个 bean 是一个 `ItemReader`，能够从数据库中分页地读取记录。

它执行 `setQueryId` 属性指定的查询来获取请求的数据。这个查询使用 `setPageSize` 属性指定了分页请求的大小，并被执行。其它的页面将在必要时被请求（例如调用 `read()` 方法时），返回对应位置上的对象。

reader 还提供了一些标准的请求参数。在命名查询的 SQL 中，必须使用部分或全部的参数（视乎 SQL 方言而定）来构造指定大小的结果集。这些参数是：

- `_page`： 欲读取的页码（从 0 开始）
- `_pagesize`： 每一页的大小，也就是返回的行数
- `_skiprows`： `_page` 和 `_pagesize` 的乘积

它们可以被映射成如下的 select 语句：

```
<select id="getEmployee" resultMap="employeeBatchResult">
  SELECT id, name, job FROM employees ORDER BY id ASC LIMIT #{_skiprows}, #{_pagesize}
</select>
```

配合如下的配置样例：

```
<bean id="reader" class="org.mybatis.spring.batch.MyBatisPagingItemReader">
  <property name="sqlSessionFactory" ref="sqlSessionFactory" />
  <property name="queryId" value="com.my.name.space.batch.EmployeeMapper.getEmployee" />
</bean>
```

```
@Configuration
public class BatchAppConfig {
  @Bean
  public MyBatisPagingItemReader<Employee> reader() {
    return new MyBatisPagingItemReaderBuilder<Employee>()
        .sqlSessionFactory(sqlSessionFactory())
        .queryId("com.my.name.space.batch.EmployeeMapper.getEmployee")
        .build();
  }
}
```

让我们通过一个更复杂一点的例子来阐明一切：

```
<bean id="dateBasedCriteriaReader"
  class="org.mybatis.spring.batch.MyBatisPagingItemReader"
  p:sqlSessionFactory-ref="batchReadingSessionFactory"
  p:parameterValues-ref="datesParameters"
  p:queryId="com.my.name.space.batch.ExampleMapper.queryUserInteractionsOnSpecificTimeSlot"
  p:pageSize="200"
  scope="step"/>

<util:map id="datesParameters" scope="step">
  <entry key="yesterday" value="#{jobExecutionContext['EXTRACTION_START_DATE']}"/>
  <entry key="today" value="#{jobExecutionContext['TODAY_DATE']}"/>
  <entry key="first_day_of_the_month" value="#{jobExecutionContext['FIRST_DAY_OF_THE_MONTH_DATE']}"/>
  <entry key="first_day_of_the_previous_month" value="#{jobExecutionContext['FIRST_DAY_OF_THE_PREVIOUS_MONTH_DATE']}"/>
</util:map>
```

```
@Configuration
public class BatchAppConfig {
  @StepScope
  @Bean
  public MyBatisPagingItemReader<User> dateBasedCriteriaReader(
      @Value("#{@datesParameters}") Map<String, Object> datesParameters) throws Exception {
    return new MyBatisPagingItemReaderBuilder<User>()
        .sqlSessionFactory(batchReadingSessionFactory())
        .queryId("com.my.name.space.batch.ExampleMapper.queryUserInteractionsOnSpecificTimeSlot")
        .parameterValues(datesParameters)
        .pageSize(200)
        .build();
  }

  @StepScope
  @Bean
  public Map<String, Object> datesParameters(
      @Value("#{jobExecutionContext['EXTRACTION_START_DATE']}") LocalDate yesterday,
      @Value("#{jobExecutionContext['TODAY_DATE']}") LocalDate today,
      @Value("#{jobExecutionContext['FIRST_DAY_OF_THE_MONTH_DATE']}") LocalDate firstDayOfTheMonth,
      @Value("#{jobExecutionContext['FIRST_DAY_OF_THE_PREVIOUS_MONTH_DATE']}") LocalDate firstDayOfThePreviousMonth) {
    Map<String, Object> map = new HashMap<>();
    map.put("yesterday", yesterday);
    map.put("today", today);
    map.put("first_day_of_the_month", firstDayOfTheMonth);
    map.put("first_day_of_the_previous_month", firstDayOfThePreviousMonth);
    return map;
  }
}
```

上面的样例使用了几个东西：

- `sqlSessionFactory`：可以为 reader 指定你自定义的 sessionFactory，当你想从多个数据库中读取数据时尤其有用
- `queryId`：指定在检索已预时要执行的查询的 ID，可以指定短的 ID 或是带命名空间的完整 ID。一般来说，你的应用可能从多个类或数据库中读取数据，因此会配置多个查询，可能会使用到在不同命名空间中的不同映射器。因此最好提供映射器文件的命名空间以便准确指定你想使用的查询 ID。
- `parameterValues`：可以通过这个 map 传递多个附加的参数，上面的例子中就使用了一个由 Spring 构建的 map，并使用 SpEL 表达式从 `jobExecutionContext` 中获取信息。而 map 的键将在映射器文件中被 MyBatis 使用（例如：`yesterday` 可以通过 `#{yesterday,jdbcType=TIMESTAMP}` 来读取）。注意，map 和 reader 都构建于 `step` 作用域，这样才能够在 Spring 表达式语言中使用 `jobExecutionContext`。另外，如果正确配置了 MyBatis 的类型处理器，你可以将自定义的实例像参数那样传递到 map 中，比如将参数类型换成 JodaTime。
- `pageSize`：如果批处理远配置了块大小（chunk size），需要通过此属性将块大小告知 reader。

MyBatisCursorItemReader

这个 bean 是一个 `ItemReader`，能够通过游标从数据库中读取记录。

提示 为了使用这个 bean，你需要使用 MyBatis 3.4.0 或更新的版本。

它执行 `setQueryId` 属性指定的查询来获取请求的数据（通过 `selectCursor()` 方法）。每次调用 `read()` 方法时，将会返回游标指向的下个元素，直到没有剩余的元素了。

这个 reader 将会使用一个单独的数据库连接，因此 select 语句将不会参与到 step 处理中创建的任何事务。

当使用游标时，只需要执行普通的查询：

```
<select id="getEmployee" resultMap="employeeBatchResult">
  SELECT id, name, job FROM employees ORDER BY id ASC
</select>
```

搭配以下的配置样例：

```
<bean id="reader" class="org.mybatis.spring.batch.MyBatisCursorItemReader">
  <property name="sqlSessionFactory" ref="sqlSessionFactory" />
  <property name="queryId" value="com.my.name.space.batch.EmployeeMapper.getEmployee" />
</bean>
```

```
@Configuration
public class BatchAppConfig {
  @Bean
  public MyBatisCursorItemReader<Employee> reader() {
    return new MyBatisCursorItemReaderBuilder<Employee>()
        .sqlSessionFactory(sqlSessionFactory())
        .queryId("com.my.name.space.batch.EmployeeMapper.getEmployee")
        .build();
  }
}
```

MyBatisBatchItemWriter

这是一个 `ItemWriter`，通过利用 `SqlSessionTemplate` 中的批量处理功能来对给定的所有记录执行多个语句。`SqlSessionFactory` 需要被配置为 `BATCH` 执行类型。

当调用 `write()` 时，将会执行 `statementId` 属性中指定的映射语句。一般情况下，`write()` 应该在一个事务中进行调用。

下面是一个配置样例：

```
<bean id="writer" class="org.mybatis.spring.batch.MyBatisBatchItemWriter">
  <property name="sqlSessionFactory" ref="sqlSessionFactory" />
  <property name="statementId" value="com.my.name.space.batch.EmployeeMapper.updateEmployee" />
</bean>
```

```
@Configuration
public class BatchAppConfig {
  @Bean
  public MyBatisBatchItemWriter<User> writer() {
    return new MyBatisBatchItemWriterBuilder<User>()
        .sqlSessionFactory(sqlSessionFactory())
        .statementId("com.my.name.space.batch.EmployeeMapper.updateEmployee")
        .build();
  }
}
```

将 ItemReader 读取的记录转换为任意的参数对象：

默认情况下，`MyBatisBatchItemWriter` 会将 `ItemReader` 读取的对象（或 `ItemProcessor` 转换过的对象）以参数对象的形式传递给 MyBatis（`SqlSession#update()`）。如果你想自定义传递给 MyBatis 的参数对象，可以使用 `itemToParameterConverter` 选项。使用该选项后，可以传递任意对象给 MyBatis。举个例子：

首先，创建一个自定义的转换器类（或工厂方法）。以下例子使用了工厂方法。

```
public class ItemToParameterMapConverters {
  public static <T> Converter<T, Map<String, Object>> createItemToParameterMapConverter(String operationBy, LocalDateTime operationAt) {
    return item -> {
      Map<String, Object> parameter = new HashMap<>();
      parameter.put("item", item);
      parameter.put("operationBy", operationBy);
      parameter.put("operationAt", operationAt);
      return parameter;
    };
  }
}
```

接下来，编写 SQL 映射。

```
<select id="createPerson" resultType="org.mybatis.spring.sample.domain.Person">
    insert into persons (first_name, last_name, operation_by, operation_at)
        values(#{item.firstName}, #{item.lastName}, #{operationBy}, #{operationAt})
</select>
```

最后，配置 `MyBatisBatchItemWriter`。

```
@Configuration
public class BatchAppConfig {
  @Bean
  public MyBatisBatchItemWriter<Person> writer() throws Exception {
    return new MyBatisBatchItemWriterBuilder<Person>()
        .sqlSessionFactory(sqlSessionFactory())
        .statementId("org.mybatis.spring.sample.mapper.PersonMapper.createPerson")
        .itemToParameterConverter(createItemToParameterMapConverter("batch_java_config_user", LocalDateTime.now()))
        .build();
  }
}
```

```
<bean id="writer" class="org.mybatis.spring.batch.MyBatisBatchItemWriter">
  <property name="sqlSessionFactory" ref="sqlSessionFactory"/>
  <property name="statementId" value="org.mybatis.spring.sample.mapper.PersonMapper.createPerson"/>
  <property name="itemToParameterConverter">
    <bean class="org.mybatis.spring.sample.config.SampleJobConfig" factory-method="createItemToParameterMapConverter">
      <constructor-arg type="java.lang.String" value="batch_xml_config_user"/>
      <constructor-arg type="java.time.LocalDateTime" value="#{T(java.time.LocalDateTime).now()}"/>
    </bean>
  </property>
</bean>
```

使用复合 writer 对多个表进行写入（但带有问题）：

这个小技巧只能在 MyBatis 3.2+ 以上的版本中使用，因为之前的版本中含有导致 writer 行为不正确的 问题 。

如果批量处理时需要写入复杂的数据，例如含有关联的记录，甚至是向多个数据库写入数据，你可能就需要一种办法来绕开 insert 语句只能插入到一个表中的限制。为了绕过此限制，批处理必须准备好要通过 writer 写入的 *成*。然而，基于对被处理的数据的观察，可以尝试使用下面的方法来解决此问题。下面的方法能够工作于具有简单关联或不相关的多个表的项。

在这种方法中，处理 Spring Batch 项的处理器中将会 *持有* 各种不同的记录。假设每个项都有一个与 *InteractionMetadata* 相关联的 *Interaction*，并且还有两个不相关的行 *VisitorInteraction* 和 *CustomerInteraction*，这时候持有器（holder）看起来像这样：

```
public class InteractionRecordToWriteInMultipleTables {
  private final VisitorInteraction visitorInteraction;
  private final CustomerInteraction customerInteraction;
  private final Interaction interaction;
  // ...
}
```

```
public class Interaction {
  private final InteractionMetadata interactionMetadata;
}
```

在 Spring 配置中要配置一个 `CompositeItemWriter`，它将会将写入操作委托到特定种类的 writer 上面去。注意 *InteractionMetadata* 在例子里面是一个关联，它需要首先被写入，这样 Interaction 才能获得更新之后的键。

```
<bean id="interactionsItemWriter" class="org.springframework.batch.item.support.CompositeItemWriter">
  <property name="delegates">
    <list>
      <ref bean="visitorInteractionsWriter"/>
      <ref bean="customerInteractionsWriter"/>

      <!-- 顺序很重要 -->
      <ref bean="interactionMetadataWriter"/>
      <ref bean="interactionWriter"/>
    </list>
  </property>
</bean>
```

```
@Configuration
public class BatchAppConfig {
  @Bean
  public CompositeItemWriter<?> interactionsItemWriter() {
    CompositeItemWriter compositeItemWriter = new CompositeItemWriter();
    List<ItemWriter<?>> writers = new ArrayList<>(4);
    writers.add(visitorInteractionsWriter());
    writers.add(customerInteractionsWriter());
    writers.add(interactionMetadataWriter());
    writers.add(interactionWriter());
    compositeItemWriter.setDelegates(writers);
    return compositeItemWriter;
  }
}
```

接下来需要配置每一个被委托的 writer；例如 *Interaction* 和 *InteractionMetadata* 对应的 writer。

```
<bean id="interactionMetadataWriter"
  class="org.mybatis.spring.batch.MyBatisBatchItemWriter"
  p:sqlSessionTemplate-ref="batchSessionTemplate"
  p:statementId="com.my.name.space.batch.InteractionRecordToWriteInMultipleTablesMapper.insertInteractionMetadata"/>
```

```
<bean id="interactionWriter"
  class="org.mybatis.spring.batch.MyBatisBatchItemWriter"
  p:sqlSessionTemplate-ref="batchSessionTemplate"
  p:statementId="com.my.name.space.batch.InteractionRecordToWriteInMultipleTablesMapper.insertInteraction"/>
```

和 reader 中的一样，通过 `statementId` 属性指定对应命名空间前缀的查询。

而在映射器配置文件中，应该根据每种特定的记录编写特定的语句，如下所示：

```
<insert id="insertInteractionMetadata"
  parameterType="com.my.batch.interactions.item.InteractionRecordToWriteInMultipleTables"
  useGeneratedKeys="true"
  keyProperty="interaction.interactionMetadata.id"
  keyColumn="id">
  <!-- 此 insert 语句使用了 #{interaction.interactionMetadata.property,jdbcType=...} -->
</insert>
```

```
<insert id="insertInteraction"
  parameterType="com.my.batch.interactions.item.InteractionRecordToWriteInMultipleTables"
  useGeneratedKeys="true"
  keyProperty="interaction.id"
  keyColumn="id">
  <!--
   此 insert 语句对普通的属性使用的是 #{interaction.property,jdbcType=...}
   而对于 InteractionMetadata 属性使用的是 #{interaction.interactionMetadata.property,jdbcType=...}
  -->
</insert>
```

执行的时候会怎么样呢？首先，`insertInteractionMetadata` 将会被调用，update 语句被设置为返回由 JDBC 驱动返回的主键（参考 `keyProperty` 和 `keyColumn`）。由于 `InteractionMetadata` 的对象被此查询更新了，下一个查询将可以通过 `insertInteraction` 开展父对象 `Interaction` 的写入工作。

然而要注意，JDBC 驱动在这方面的行为并不总是与此相一致。在编写文档时，H2 的数据库驱动 1.3.168 甚至只在 BATCH 模式下返回最后的索引值（参考 `org.h2.jdbc.JdbcStatement#getGeneratedKeys`），而 MySQL 的 JDBC 驱动则工作良好并返回所有 ID。