Introduction to Spring Batch

- Batch Application Concepts
- Spring Batch Concepts
- Writing and Configuring a Spring Batch Application

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Batch Applications

- Batch applications are different than interactive/GUI applications:
    - Don't require user input
    - Often run in "off time" at scheduled times
    - Read input from databases, flat files, XML, etc.
    - Perform dataprocessing and generate output
    - Often require transactions (units of work)
    - Often require multiple parallel execution units (threads)

Batch processing is perhaps the oldest form of business computing, and has traditionally been run on mainframes.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Writing Java Batch Applications

- Java supports non-interactive programs that can read/write files and so forth as well as threading, XML support and so forth
- However, Java provides little support for scheduling batch execution or for automatically retrying failed executions
- There are third-party frameworks such as **Quartz** for scheduling

You can read more about Quartz at http://quartz-scheduler.org.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to Spring Batch

- **Spring Batch** is a Spring subproject that lets you work with Spring batch applications
- Spring Batch integrates with schedulers such as **Quartz** or Unix **cron** or Spring 3.x native scheduler
- Spring Batch provides functionality such as logging, transaction management, job statistics, job restart on error and so forth

Spring Batch is a collaborative venture of the Spring Source organization and Accenture.

Spring Batch version 2 is a significant rewrite of the framework to take advantage of Java 5 generics and other features. It also supports an enhanced workflow model where steps can be conditional (like a Java "if" statement).

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Spring Batch Fundamentals

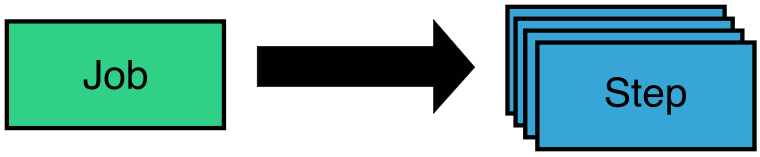
- A **job** consists of one or more **steps**
- **JobParameters** allow you to pass information to a job
- The combination of a **job** and **JobParameters** is referred to as a **JobInstance**
- Each run of a JobInstance is referred to as a **JobExecution**
- A **JobRepository** is a persistent object that aggregates all of the other Spring Batch objects

Note that there may be more than one JobExecution associated with a JobInstance if the framework has to restart (retry) the job because of errors.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Jobs

- A **job** is a restartable unit consisting of **steps**
- You don't usually need to code a Job class - just use and configure one of the provided ones such as **SimpleJob**

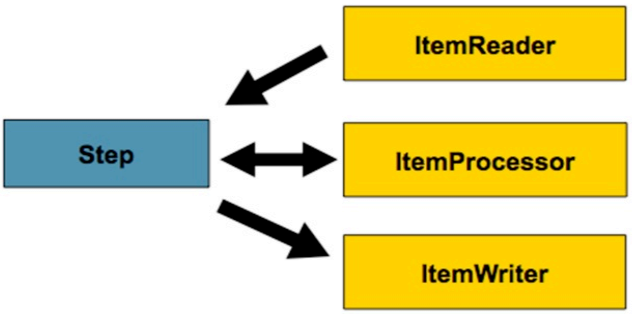


When you configure a Job, you can assign an "incrementer" that assigns Job IDs in some fashion. For example, you can use the provided RunIdIncrementer that uses simple sequence numbers.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Steps

- Steps comprise **ItemReaders**, **ItemProcessors** and **ItemWriters**
- You can use **chunks** to specify how many operations to group in a single transaction before committing changes



Spring Batch can use "chunks" to optimize output. Essentially, you specify a chunk size, and Batch queues up writes until that many items have been read and processed. Batch then writes and commits the chunk as unit - e.g. a single database transaction.

ItemProcessors are optional and provide a way for applications to process or transform the input data in some way, for example to change user-entered five-digit ZIP codes to standard postal service codes.

Spring Batch also provides several utility classes for working with I/O including LineMappers that help you work with line-oriented files, mapping lines to domain objects.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Item Readers and Writers

- Spring Batch provides several item reader and writer classes, including:
  - Flat file
  - JDBC
  - JPA/Hibernate
  - JMS
- You can write custom readers or writers by implementing the **ItemReader** and/or **ItemWriter** interfaces

---

For a complete listing of provided classes, see the Javadoc for the ItemReader and ItemWriter interfaces.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JobExecution Objects

- A JobExecution object represents an attempt to run a **Job**
- A JobExecution bean defines standard properties including:
    - **status** - BatchStatus enum: STARTED, COMPLETED, FAILED ...
    - **startTime** - The java.util.Date start time
    - **endTime** - The java.util.Date end time
    - **readCount** - Number of items read
    - **writeCount** - Number of items written

JobExecution objects are typically persisted in a database so you can query them after executions.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The JobRepository

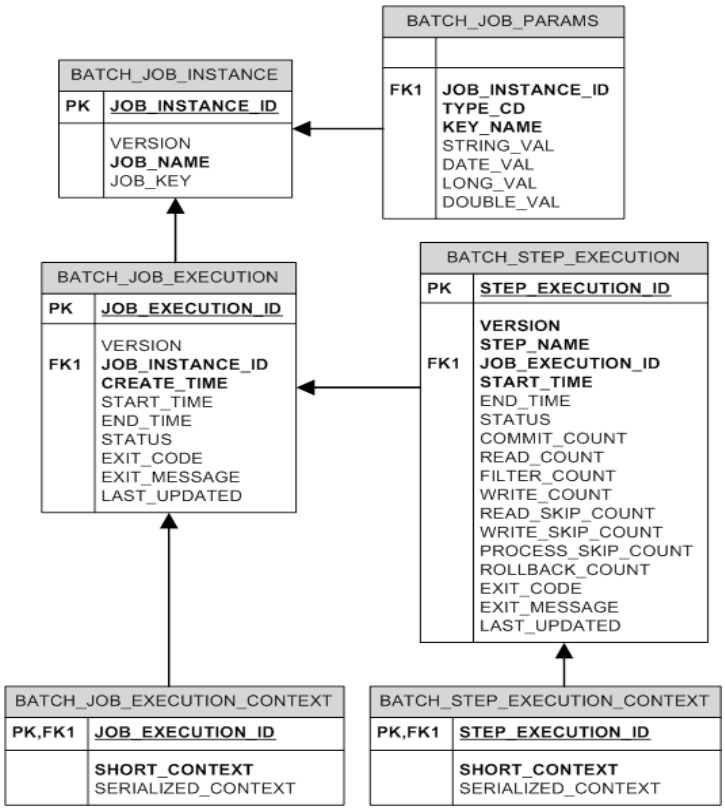
- A **JobRepository** represents the aggregation of data about job executions
- JobRepositories are responsible for storing objects (jobs, steps and so forth) in a database
- While you don't need to deal with JobRepository objects in code, you do need to configure one as a Spring bean

The JobRepository persistently stores information about jobs and executions.

This means that you MUST configure a database, even if your batch application itself doesn't use a database (batch apps invariably work with data, however).

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JobRepository Database Tables



Your database must define the tables for the JobRepository. Spring Batch provides SQL scripts for popular databases to create the tables.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Using Spring Batch in a Standalone Application

- Configure a JobRepository bean and its database connection
- Determine input source and configure appropriate **ItemReader** bean (either standard or custom)
- Determine output destination and configure appropriate **ItemWriter** bean
- Write a **main** method that executes the job

- You can configure Spring Batch using either XML or Java configuration

Note that you don't have to run Spring Batch applications from a main() method - you could run them from a servlet, for example, using an asynchronous approach.

Also note that it's common to schedule executions using a scheduler like Quartz.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Configuring JobRepository in XML

```
1    <bean id="jobRegistry"
2      class="org.spring...batch...MapJobRegistry" />
3    <bean
4      class="org.spring...JobRegistryBeanPostProcessor">
5      <property name="jobRegistry"
6         value="jobRegistry" />
7    </bean>
8    <bean id="jobLauncher"
9      class="org.spring...SimpleJobLauncher">
10     <property name="jobRepository"
11        value="jobRepository" />
12   </bean>
13   <bean id="jobRepository"
14      class="org.spring...JobRepositoryFactoryBean">
15     <property name="dataSource" value="myDS" />
16     <property name=""transactionManager" value="myTM" />
17   </bean>
```


We have elided the class names so they fit on the page. The actual fully qualified names are:

```
org.springframework.batch.core.configuration.support.MapJobRegistry
org.springframework.batch.core.launch.support.SimpleJobLauncher
org.springframework.batch.core.configuration.support.JobRegistryBeanPostProcessor
org.springframework.batch.core.repository.support.JobRepositoryFactoryBean
```

Note that we also need to configure a standard Spring data source and transaction manager; those beans are not shown here.

The MapJobRegistry bean maintains the list of jobs - the JobRegistryBeanPostProcessor scans to find configured jobs and adds them to the registry.

The SimpleJobLauncher provides a run() method that accepts the job parameters.

Finally, the jobRepository bean stores the job metadata to the database specified by the dataSource using the specified transaction manager.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Configuring a Job in XML

- Here we configure a job that uses an ItemReader that can parse CSV files and an ItemWriter that inserts into a database

```
1    <job
2      job-repository="jobRepository"
3      id="myJob">
4      <step id="step1">
5        <tasklet
6           transaction-manager="transactionManager">
7          <chunk commit-interval="5"
8            reader="csvFileReader"
9            writer="jdbcItemWriter"/>
10       </tasklet>
11     </step>
12   </job>
```

Spring Batch provides a special XML namespace that you can use to configure jobs and other beans. Here we use it to define a job with a single step. Note that we would need to define the cvsReader and jdbcItemWriter beans - they are not shown in this abreviated listing. The <job> element configures a SimpleJob object.

If you have multiple steps, you must chain them together using the "next" attribute, for example:

```
<step id="initialize_report" next="process">
. . .
</step>
<step id="process" >
. . .
</step>
```

Note that the first step must always appear first in the configuration, after that, the "next" attribute controls ordering.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Writing a main Method

```
1    public static void main(String[] args)
2    {
3      ClassPathXmlApplicationContext ctx =
4        new ClassPathXmlApplicationContext("spring.xml");
5      JobLauncher jobLauncher = (JobLauncher)
6        ctx.getBean("jobLauncher");
7      Job job = (Job) ctx.getBean("myJob");
8      JobExecution jobExecution =
9        jobLauncher.run(job, new JobParameters());
10
11     BatchStatus batchStatus = jobExecution.getStatus();
12     while(batchStatus.isRunning())
13     {
14       System.out.print("."); Thread.sleep(10000);
15     }
16     System.out.println("Done: " + batchStatus);
17   }
```


Here, we launch the batch from a Java standalone application.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Spring Batch with Spring Boot

- Spring Boot simplifies getting started with Spring Batch:
  - Provides a **starter dependency** that brings in all Spring Batch libraries and their dependencies
  - Automatically configures **factories** for creating jobs and steps
  - Automatically configures an in-memory database for the JobRepository - useful for getting started
- Spring Boot apps typically use Java configuration instead of XML

BatchConfiguration.java

Of course, you can configure a "real" persistent database for production.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
1   package com.example;
2
3   import org.apache.log4j.Logger;
4   import org.springframework.batch.core.Job;
5   import org.springframework.batch.core.Step;
6   import org.springframework.batch.core.StepContribution;
7   import org.springframework.batch.core.configuration.annotation.EnableBatchProcessi
8   import org.springframework.batch.core.configuration.annotation.JobBuilderFactory;
9   import org.springframework.batch.core.configuration.annotation.StepBuilderFactory;
10  import org.springframework.batch.core.launch.support.RunIdIncrementer;
11  import org.springframework.batch.core.scope.context.ChunkContext;
12  import org.springframework.batch.core.step.tasklet.Tasklet;
13  import org.springframework.batch.repeat.RepeatStatus;
14  import org.springframework.beans.factory.annotation.Autowired;
15  import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
16  import org.springframework.context.annotation.Bean;
17  import org.springframework.context.annotation.Configuration;
18
19  @Configuration
20  @EnableBatchProcessing
21  @EnableAutoConfiguration
22  public class BatchConfiguration
23  {
24
25      private Logger log = Logger.getLogger(BatchConfiguration.class);
26
27
28      @Autowired
29      private JobBuilderFactory jobBuilderFactory;
30
31      @Autowired
32      private StepBuilderFactory stepBuilderFactory;
33
34      @Bean
35      public Step step1()
36      {
37          return stepBuilderFactory.get("step1").tasklet(new Tasklet()
38          {
39              public RepeatStatus execute(StepContribution contribution,
40                      ChunkContext chunkContext)
41              {
42                  log.info("************* STEP 1");
43                  return RepeatStatus.FINISHED;
44              }
45          }).build();
```

2 - 16A : BatchConfiguration.java

```
46      }
47
48      @Bean
49      public Job job() throws Exception
50      {
51          return
52            jobBuilderFactory.get("job1").incrementer(new RunIdIncrementer())
53                  .start(step1()).build();
54      }
55  }
```

2 - 16B : BatchConfiguration.java

Chapter Summary

In this chapter, you learned:

- The basics of the Spring Batch project

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum