Spring Batch

Lesson 01 : Introduction to Spring Batch

Lesson Objectives

- Introduction to batch processing
- Introduction to Spring Batch
- Spring Batch architecture
- Spring Batch concepts
- Example : Spring Batch Hello World
- Passing Job Parameters

Batch Processing

- **Most of the applications you develop have an aspect of user interaction, whether it's a user clicking a link in a web app, typing information into a form on a thick client, or tapping around on phone and tablet apps.**
- **Batch processing is the exact opposite of those types of applications.**
  - Batch processing is defined as the processing of data (large amounts) without interaction or interruption. Once started, a batch process runs to some form of completion without any intervention.
  - Batch processing solutions typically run offline
  - Exchanging data, computing data, generating monthly financial statements, calculating statistics, indexing files are some examples of batch applications.

Why do we need batch processing?

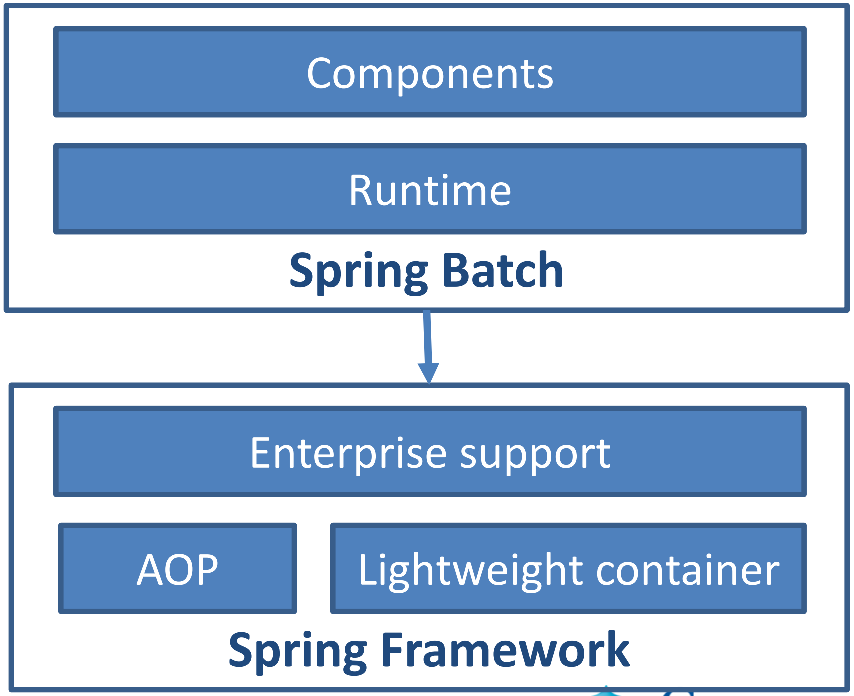
- **You don't always have all the required information immediately.**
  - Batch processing allows you to collect information required for a given process before starting the required processing.
- **Sometimes it makes good business sense.**
  - Batch processing is used to process billions of transactions everyday within mission-critical enterprise applications.
- **It can be a better use of resources.**
  - Having a lot of processing power sitting idle is expensive. It's more cost effective to have a collection of scheduled processes that run one after the other using the machine's full potential at a constant, predictable rate.

Spring Batch : Introduction

- **Spring Batch is an open source framework for batch processing – project was started in 2007.**
  - It is a lightweight, comprehensive solution designed to enable the development of robust batch applications, which are often found in modern enterprise systems.
  - Spring Batch builds upon the POJO-based development approach of the Spring Framework
- **Spring Batch provides reusable functions that are essential in processing large volumes of records, including logging/tracing, transaction management, job processing statistics, job restart, skip, and resource management.**
- **Features implemented by Spring Batch include data validation, formatting of output, the ability to implement complex business rules in a reusable way, and the ability to handle large data sets.**

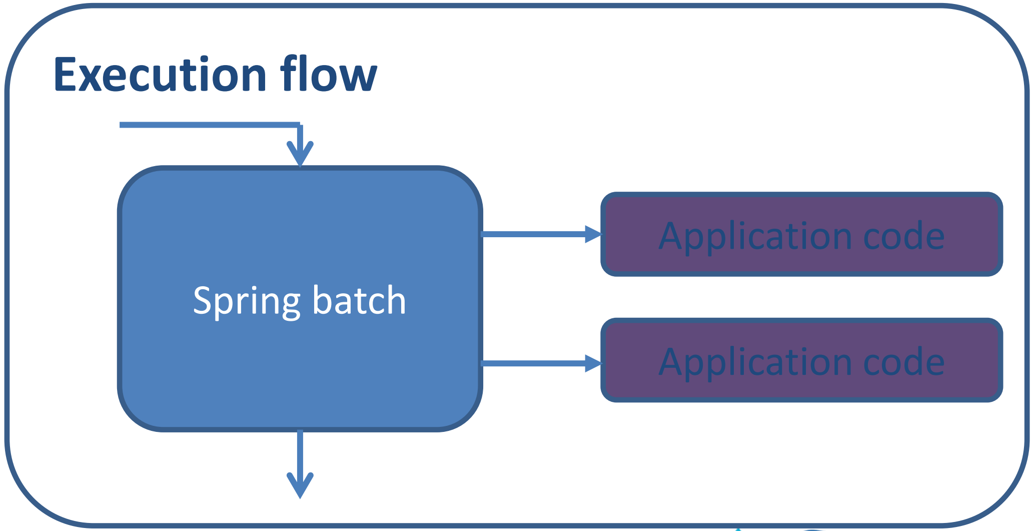
Spring Batch

- **Spring Batch builds on top the Spring Framework, so it can leverage :**
  - its lightweight container for configuration
    - All Spring Batch's components are meant to be configured by Spring's lightweight container, leveraging dependency injection and some common hooks (complex object instantiation, initialization callbacks, dedicated scope)
  - the aspect-oriented programming framework to address cross-cutting
  - the enterprise support to integrate with enterprise systems like databases



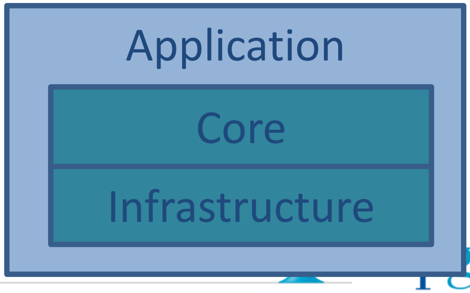
Batch-oriented runtime

- **Batch-oriented runtime refers to the way Spring Batch can drive the flow of a batch process.**
  - Once you use Spring Batch, it takes charge of orchestrating the flow of your batch application: when & how to read records from the database, when to open a stream to file, when to commit the transaction etc
  - At some places in this flow, eg inside a transaction, Spring Batch will call your own code to perform the core business operation.
  - Figure shows how Spring Batch drives the application flow and calls business code appropriately.



Spring Batch architecture

- **Spring Batch was designed as a three layered architecture**
  - **application layer** : consists of all the custom code and configuration used to build out your batch processes.
    - Your business logic, services, and so on, as well as the configuration of how you structure your jobs, are all considered the application.
  - **core layer** : contains pieces that define the batch domain.
    - Elements of the core component include the Job and Step interfaces & the interfaces used to execute a Job: JobLauncher & JobParameters.
  - **infrastructure layer** : In order to do any processing, you need to read and write from files, databases, and so on. You must be able to handle what to do when a job is retried after a failure.
    - These pieces are considered common infrastructure and live in the infrastructure component of the framework.



Spring Batch concepts

- Job is the main component of Spring Batch & represents a batch process, which is typically made up of a series of Steps.
- A Step is an independent process of a batch Job that contains all of the information necessary to define and control a particular phase in the job execution.
- The Step may contain a single Tasklet that is used for simple processing such as validating job parameters when launching a job, setting up various resources, cleaning up resources, etc.

```
<bean id=“accountTasklet” class="com.igate.AccountTasklet"/>
<job id="accountJob">
    <step id="accountStep">
          <tasklet ref="accountTasklet"/>
    </step>
</job>
```

Spring Batch concepts

- **JobRepository : a datastore (in memory or a database) that is used to persist information about the job and step executions**
  - Two sets of implementations are provided by Spring Batch: Map based (in-memory) and Jdbc based
- **JobLauncher : helps to launch a job.**
  - JobLaunchers are responsible for starting a Job with a given job parameters.
  - The provided implementation, SimpleJobLauncher, relies on a TaskExecutor to launch the jobs.
- **JobInstance : A running instance of a job.**
  - Think job as class and job instance as object.
- **JobParameters : Parameters that go into a JobInstance**
- **A JobExecution or StepExecution : is information about a single run of the job or step.**

Example : Spring Batch Hello World

```
public class HelloTasklet implements Tasklet{
  private String message;
  public void setMessage(String message) { this.message = message;  }
  public RepeatStatus  execute(StepContribution arg0, ChunkContext
arg1){
        System.out.print(message);
         return RepeatStatus.FINISHED;
}
```

```
<bean id="hello" class="com.igate.HelloTasklet">
     <property name="message" value="Hello! Welcome to Spring Batch!" />
</bean>
<batch:job id="helloJob">
       <batch:step id="helloStep">
              <batch:tasklet ref="hello" />
       </batch:step>
</batch:job>
```

Setting up Spring Batch's infrastructure

Spring Batch relies on some Spring beans to fulfill its infrastructure work: transaction management, storage of job executions and states, launching of jobs and so on.

this configuration must be done only once for all jobs

```
<beans …>
  <bean id="transactionManager"    class=
  "org.springframework.batch.support.transaction.ResourcelessTransactionManager"
  />
 <bean id="jobRepository”  class=
"org.springframework.batch.core.repository.support.MapJobRepositoryFactoryBean">
     <property name="transactionManager" ref="transactionManager" />
 </bean>

 <bean id="jobLauncher"
  class="org.springframework.batch.core.launch.support.SimpleJobLauncher">
     <property name="jobRepository" ref="jobRepository" />
 </bean>
</beans>
```

Launching the batch

```
public class LaunchHelloworldjob {
    public static void main(String... args) throws Exception {
        ApplicationContext context =
            new ClassPathXmlApplicationContext("simplejob.xml");
        JobLauncher jobLauncher = context.getBean(JobLauncher.class);
        Job job = context.getBean(Job.class);
        JobExecution jobExecution =
                    jobLauncher.run(job, new JobParameters());
    }
}
```

```
public class LaunchHelloworldjob {
  public static void main(String... args) throws Exception {
    CommandLineJobRunner.main(new String[]{“simplejob.xml", “helloJob"});
  }
}
```