Spring Batch Workshop

Arnaud Cogoluègnes

Consultant with Zenika, Co-author Spring Batch in Action

May 23, 2011

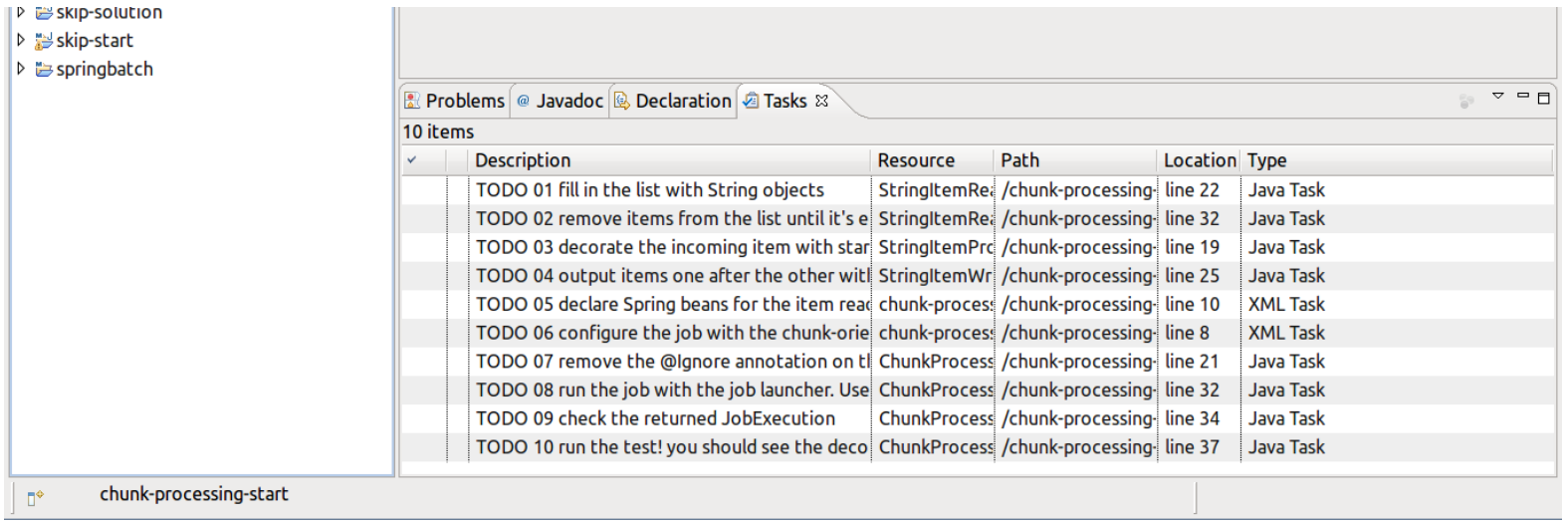
Overview

- This workshop highlights Spring Batch features
- Problem/solution approach
  - A few slides to cover the feature
  - A project to start from, just follow the TODOs
- Prerequisites
  - Basics about Java and Java EE
  - Spring: dependency injection, enterprise support
- https://github.com/acogoluegnes/Spring-Batch-Workshop

License

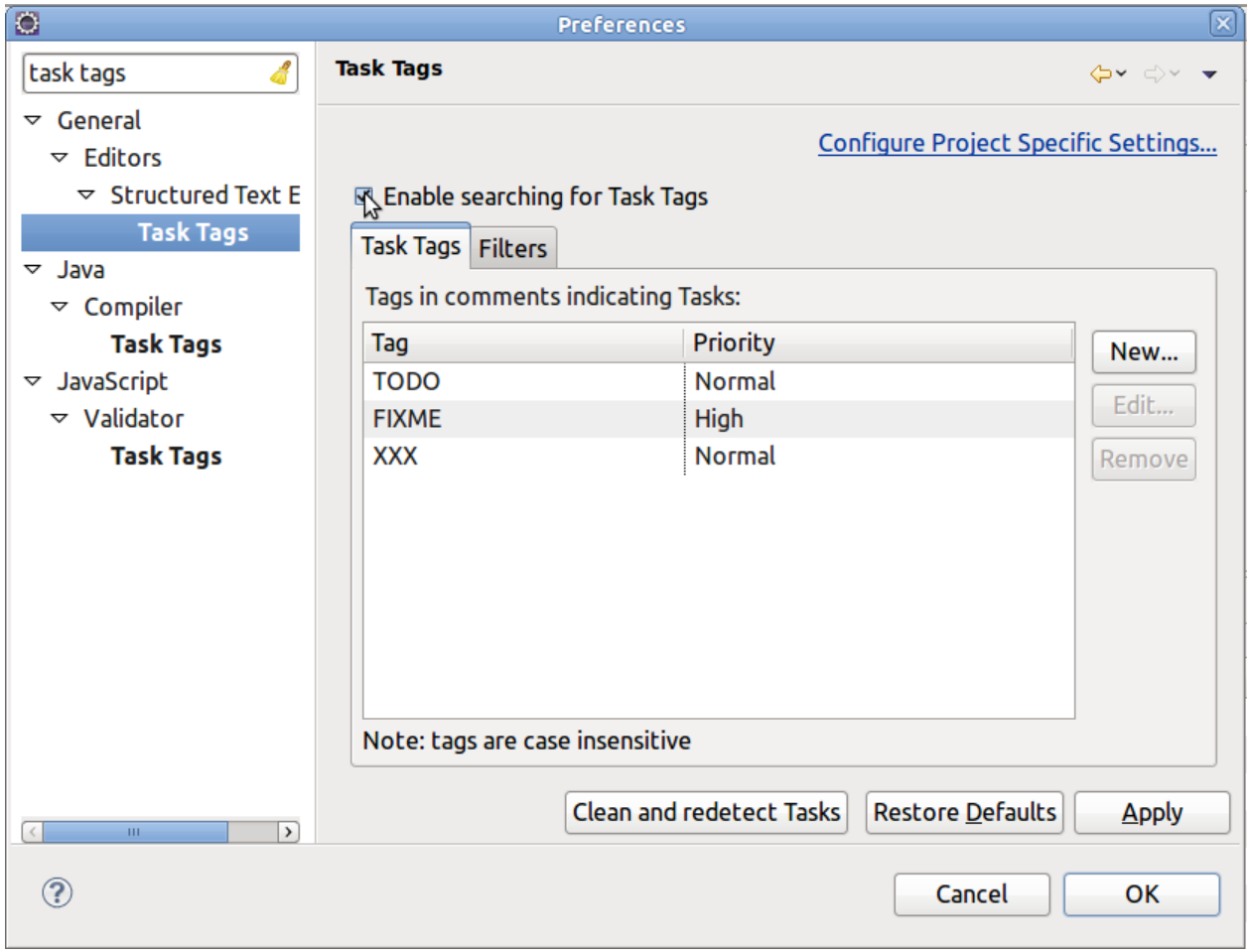
Follow the TODOs

- Track the TODO in the *-start projects!
- It's easier with support from the IDE



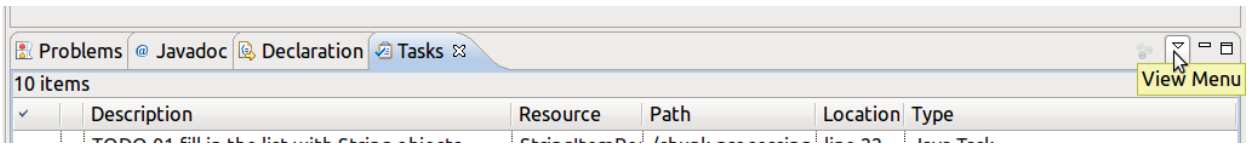
TODO with Eclipse

- Window > Preferences > “tasks tag” in filter



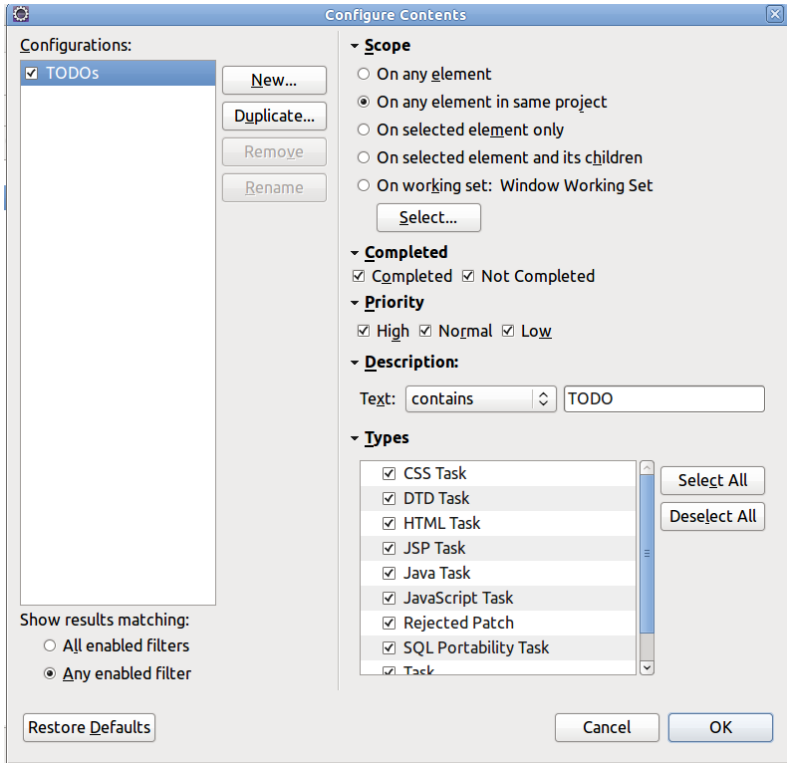
TODO with Eclipse

- Open the “Tasks” view
- click on the down arrow on the right
- “configure contents”



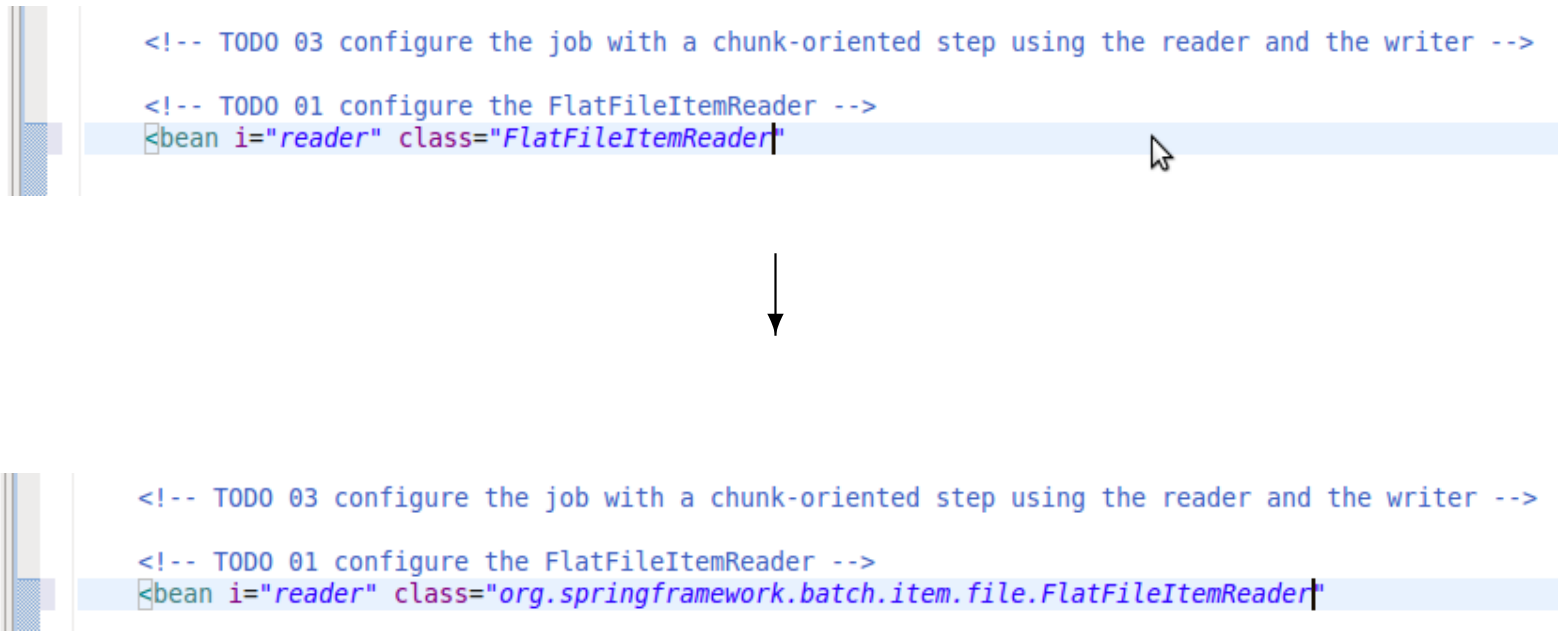
TODO with Eclipse

- Check “TODOs” on the left
- Check “On any element in the same project” on the right (scope)



Spring support in IDE is a +

- e.g. code completion in SpringSource Tool Suite



Basic features for batch applications

- Read – process – write large amounts of data, efficiently
- Ready-to-use components to read from/write to
  - Flat/XML files
  - Databases (JDBC, Hibernate, JPA, iBatis)
  - JMS queues
  - Emails
- Numerous extension points/hooks

Advanced features for batch applications

- Configuration to skip/retry items
- Execution metadata
  - Monitoring
  - Restart after failure
- Scaling strategies
  - Local/remote
  - Partitioning, remote processing

- Problem: getting started with Spring Batch
- Solution: writing a simple “Hello World” job

Structure of a job

- A Spring Batch job is made of steps
- The Hello World job has one step
- The processing is implemented in a `Tasklet`

The Hello World Tasklet

```
public class HelloWorldTasklet implements Tasklet {

  @Override
  public RepeatStatus execute(
      StepContribution contribution,
      ChunkContext chunkContext) throws Exception {
    System.out.println("Hello world!");
    return RepeatStatus.FINISHED;
  }

}
```

The configuration of the Hello World job [1]

```
<batch:job id="helloWorldJob">
  <batch:step id="helloWorldStep">
    <batch:tasklet>
      <bean class="com.zenika.workshop.springbatch.HelloWorldTasklet" />
    </batch:tasklet>
  </batch:step>
</batch:job>
```

[1] Notice the `batch` namespace

Spring Batch needs some infrastructure beans

- Let's use the typical test configuration

```
<bean id="transactionManager"
      class="o.s.b.support.transaction.ResourcelessTransactionManager" />

<bean id="jobRepository"
      class="o.s.b.core.repository.support.MapJobRepositoryFactoryBean" />

<bean id="jobLauncher"
      class="o.s.b.core.launch.support.SimpleJobLauncher">
  <property name="jobRepository" ref="jobRepository" />
</bean>
```

Running the test in a JUnit test

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("/hello-world-job.xml")
public class HelloWorldJobTest {

  @Autowired
  private Job job;

  @Autowired
  private JobLauncher jobLauncher;

  @Test public void helloWorld() throws Exception {
    JobExecution execution = jobLauncher.run(job, new JobParameters());
    assertEquals(ExitStatus.COMPLETED, execution.getExitStatus());
  }
}
```

- Problem: processing large amounts of data efficiently
- Solution: using chunk processing

What is chunk processing?

- Batch jobs often read, process, and write items
- e.g.
  - Reading items from a file
  - Then processing (converting) items
  - Writing items to a database
- Spring Batch calls this “chunk processing”
- a chunk = a set of items

Chunk processing with Spring Batch

- Spring Batch
  - handles the iteration logic
  - uses a transaction for each chunk
  - lets you choose the chunk size
  - defines interfaces for each part of the processing

The reading phase

- Spring Batch creates chunks of items by calling `read()`
- Reading ends when `read()` returns `null`

```
public interface ItemReader<T> {

  T read() throws Exception, UnexpectedInputException,
                  ParseException, NonTransientResourceException;

}
```

The processing phase

- Once a chunk is created, items are sent to the processor
- Optional

```
public interface ItemProcessor<I, O> {

  O process(I item) throws Exception;

}
```

The writing phase

- Receives all the items of the chunk
- Allows for batch update (more efficient)

```
public interface ItemWriter<T> {

  void write(List<? extends T> items) throws Exception;

}
```

An example

- Let's implement a (too?) simple chunk-oriented step!

The ItemReader

```
public class StringItemReader implements ItemReader<String> {

  private List<String> list;

  public StringItemReader() {
    this.list = new ArrayList<String>(Arrays.asList(
      "1","2","3","4","5","6","7")
    );
  }

  @Override
  public String read() throws Exception, UnexpectedInputException,
                              ParseException, NonTransientResourceException {
    return !list.isEmpty() ? list.remove(0) : null;
  }
}
```

The ItemProcessor

```
public class StringItemProcessor implements ItemProcessor<String, String> {

  @Override
  public String process(String item) throws Exception {
    return "*** "+item+" ***";
  }

}
```

The ItemWriter

```
public class StringItemWriter implements ItemWriter<String> {

  private static final Logger LOGGER =
    LoggerFactory.getLogger(StringItemWriter.class);

  @Override
  public void write(List<? extends String> items) throws Exception {
    for(String item : items) {
      LOGGER.info("writing "+item);
    }
  }

}
```

Configuring the job

```
<batch:job id="chunkProcessingJob">
  <batch:step id="chunkProcessingStep">
    <batch:tasklet>
      <batch:chunk reader="reader" processor="processor" writer="writer"
                   commit-interval="3"
       />
    </batch:tasklet>
  </batch:step>
</batch:job>

<bean id="reader" class="com.zenika.workshop.springbatch.StringItemReader" />

<bean id="processor"
      class="com.zenika.workshop.springbatch.StringItemProcessor" />

<bean id="writer" class="com.zenika.workshop.springbatch.StringItemWriter" />
```

Considerations

- Do I always need to write my `ItemReader`/`Processor`/`Writer`?
- No, Spring Batch provides ready-to-use components for common datastores
  - Flat/XML files, databases, JMS, etc.
- As an application developer, you
  - Configure these components
  - Provides some logic (e.g. mapping a line with a domain object)

Going further...

- Reader/writer implementation for flat/XML files, database, JMS
- Skipping items when something goes wrong
- Listeners to react to the chunk processing