Building For Performance with Spring Integration & Spring Batch

Andrew Elmore

andrew.elmore@incept5.com

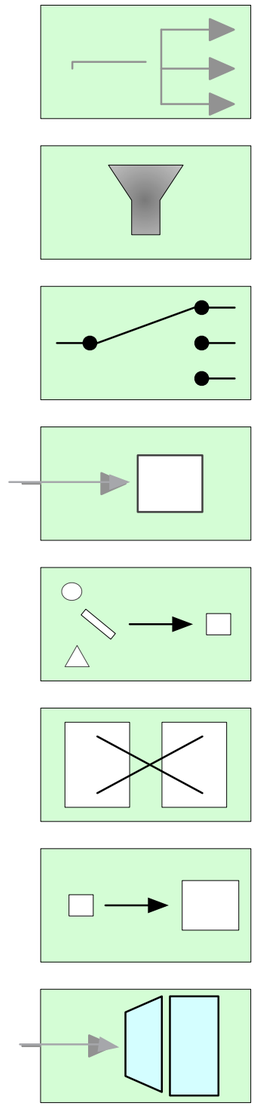
@andrew_elmore

Agenda

- What is ‘good’ performance
- How fast is Spring Integration?
- Measuring & reducing latency
- Scaling up
  - Increasing throughput
  - Handling batch messages

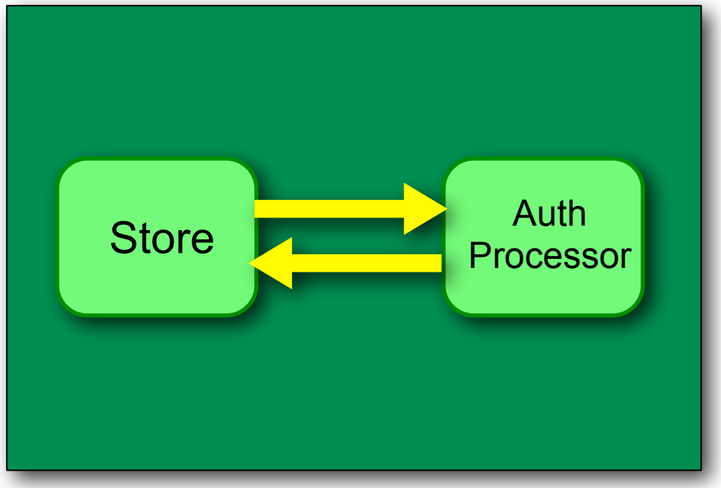
Spring Integration

- Implementation of the Enterprise Integration Patterns
- Large range of endpoint adapters
- Unlike past ESBs, can have minimal impact on application code

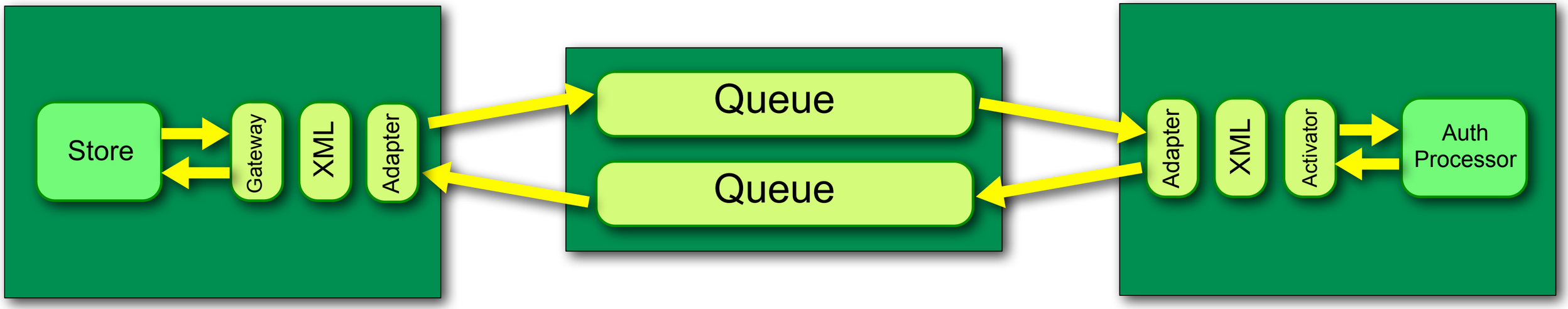


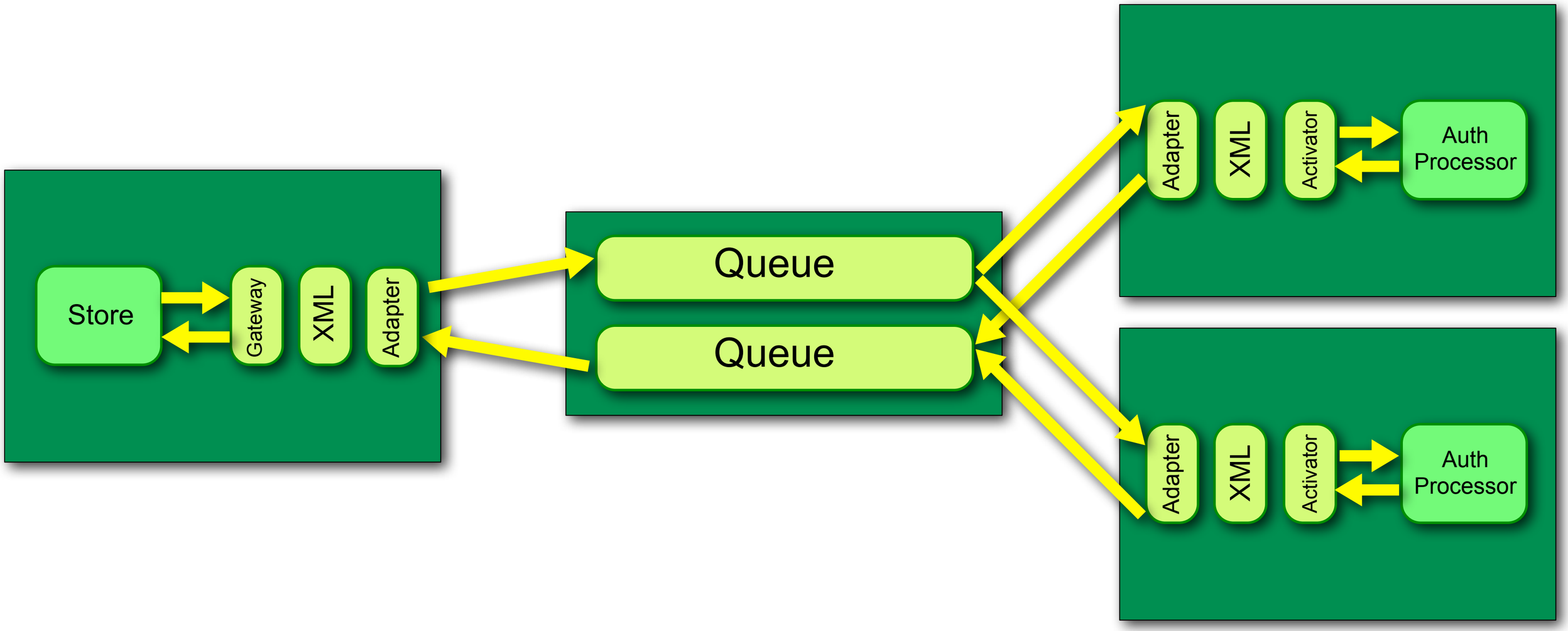
Zero-impact Code Remoting

- The Challenge -
  - Run Store & Auth Processor on separate machines
  - ...without making any changes to either class



Disclaimer - just because you can doesn't mean you should!





AMQP Outbound Gateway

- Create a private response queue
  - Configure durability etc
- Create a request message
  - Set content type, persistence, payload, reply queue, immediate delivery etc
- Publish the message to an exchange with a routing key
  - Handle transactions, publisher confirms, return listeners etc if required
- Wait for and consume the response
  - Provide necessary acknowledgements back to the broker
- Extract payload

- And of course handle all the error & retry scenarios
  - Connection dropped
  - Nacks
  - No reply
  - ...

Or Just...

```
<amqp:outbound-gateway
    request-channel="client.request.xml" reply-channel="client.response.xml"
    exchange-name="PurchaseRequests" routing-key="Europe" amqp-template="amqpTemplate"/>
```

Client Config

```
<gateway id="authService"
         service-interface="com.incept5.model.AuthorisationService"
         default-request-channel="client.request.java"
         default-reply-channel="client.response.java"/>


<chain input-channel="client.request.java" output-channel="client.request.xml">
    <int-c24:transformer transform-class="com.incept5.c24.transform.AuthRequestToXmlTransform"/>
    <int-c24:marshalling-transformer sink-factory="xmlSinkFactory" output-type="STRING"/>
</chain>

<amqp:outbound-gateway request-channel="client.request.xml" reply-channel="client.response.xml"
                       exchange-name="PurchaseRequests" routing-key="Europe" amqp-template="amqpTemplate"/>

<chain input-channel="client.response.xml" output-channel="client.response.java">
    <int-c24:unmarshalling-transformer source-factory-ref="xmlSourceFactory" model-ref="XmlResponseElement"/>
    <int-c24:transformer transform-class="com.incept5.c24.transform.XmlToAuthResponseTransform"/>
</chain>
```

Speed

The Value of a Millisecond - Financial Services

- Financial services estimate 1ms advantage worth $100M pa
  - –Hibernia Atlantic spending $300M to reduce LON<>NY by 5.2ms (59.6ms vs 64.8ms)
  - –Spread Networks buried a new 825 mile fibre cable Chicago<>NY to save 1.4ms (13.1ms vs 14.5ms)
    - Tradworx building a microwave solution (8.5ms - $250,000 pa for futures-market feed)
  - –It's all about finding a shorter path
    - You cannae change the laws of physics!

The Value of a Millisecond

- Amazon - 100ms == 1% lost sales (annual sales $42B)
- Yahoo - 400ms delay drops traffic 5-9%
- But we generally don’t just have to do one thing quickly...

So Much To Do, So Little Time

- Twitter has peaked at over 25,000 tweets / s
- Facebook > 1B hits per day (>1M /s)
- Google - 3B searches per day
- Google's Santa Tracker - peaked at over 1.6M hits/s

What Do We Typically Encounter

- Latency Requirements
  - A few ms +
  - Sub-millisecond processing tends to be C/C++
- Throughput requirements
  - Message-type dependent
  - Up-to 000,000s / s
- Processing Windows
  - Cut-off times for processing run between
- Per-message & Batch

Common Concerns

- I’m not in control of its performance
  - Overhead is low for most scenarios
    - See later...
- XML overload
  - There are other options - Java, Scala, ...

Timing Spring Integration

What Does SI Do Under The Covers?

- Creates a message to hold the payload
- Sets appropriate headers
- Routes the message between components
- Invokes behaviour on the components, which may include
  - Introspect the component
  - Invoke functionality via reflection
  - Create new message

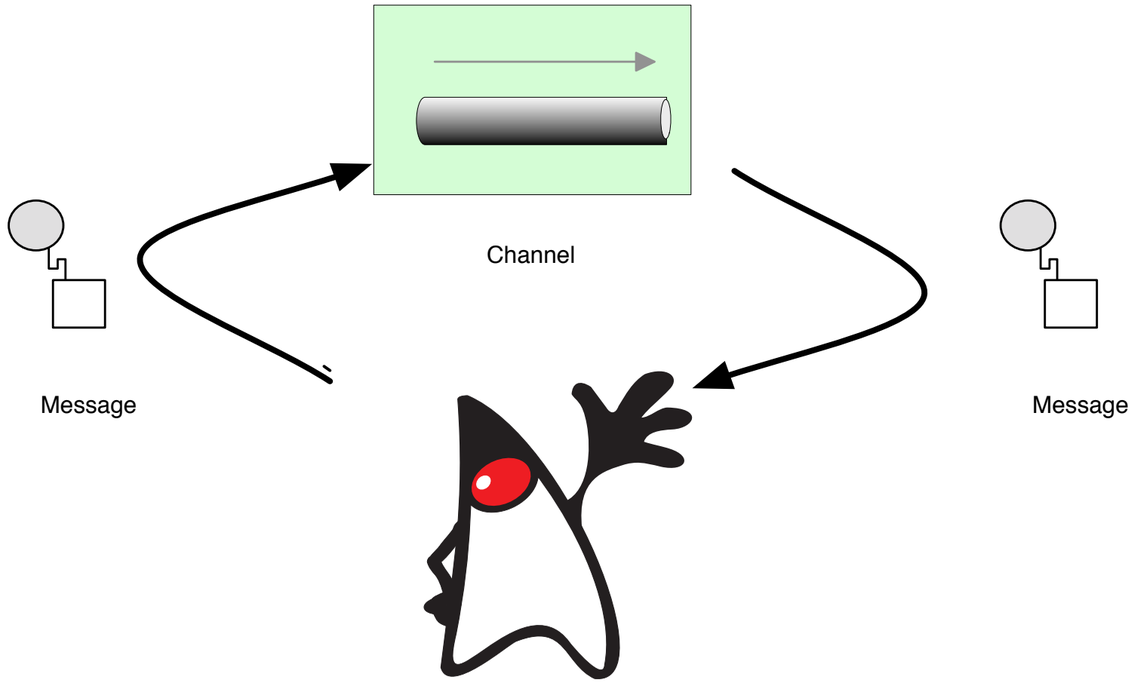
- Does this cost?
  - Yep

- How much?
  - Let's find out...

Cost of Admission

- Send message to a Queueing Channel
- Receive it from the Channel

- Running on a 2.3GHz i7



Cost of Admission

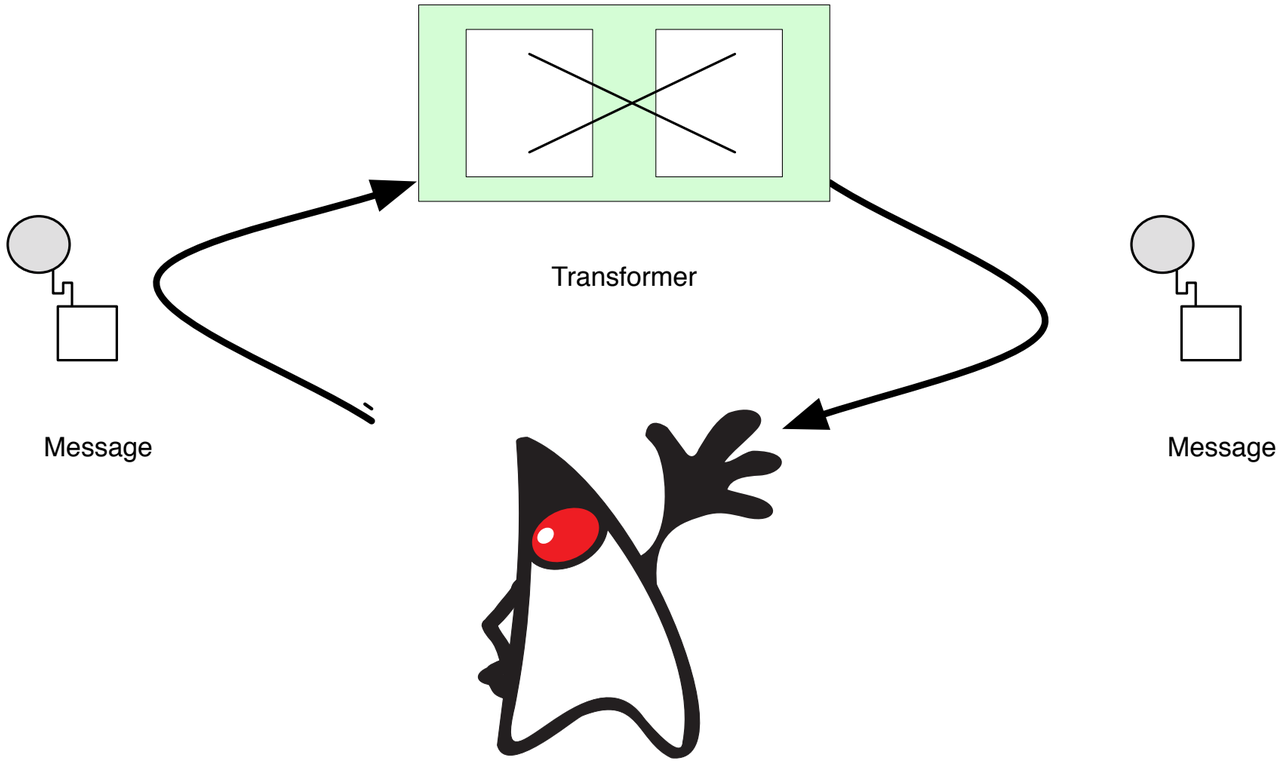
- Entry cost for using SI <4us
- Why do we see variation?
  - –Process does not have exclusive access to resources
  - –JVM jitter
- Really low latency apps don’t tend to be written in Java
  - –Determinism is important

A Failed Metaphor

- Usain Bolt
  - –Fastest man on Earth
  - –100m in 9.58s
- In 4us Usain would travel 0.4mm
- A single thread would process 2.4M messages before Bolt crossed the line

Logic Invocation Cost

- What does it cost us to expose existing logic as a transformer?



Transformer

- About 2us if we tie our code to SI

```
<beans:bean class="biz.c24.springone.components.IncrementingTransformer"/>
```

- About 4.5us if we don't

```
<beans:bean id="incrementor" class="biz.c24.springone.components.Incrementor"/>
...
<transformer method="increment" ref="incrementor"/>
```

- Standard rules apply
  - –Keep it simple - introduce complexity only where necessary

Why So Much Faster?

- Because SI has to do less work
  - –Less introspection, reflection
  - –Smaller call-stack

IncrementingTransformer.increment(int) line: 10
NativeMethodAccessorImpl.invoke0(Method, Object, Object[]) line: not available [native method]
NativeMethodAccessorImpl.invoke(Object, Object[]) line: 39
DelegatingMethodAccessorImpl.invoke(Object, Object[]) line: 25
Method.invoke(Object, Object...) line: 597
ReflectiveMethodExecutor.execute(EvaluationContext, Object, Object...) line: 69
MethodReference.getValueInternal(ExpressionState) line: 109
CompoundExpression.getValueInternal(ExpressionState) line: 57
CompoundExpression(SpelNodeImpl).getTypedValue(ExpressionState) line: 102
SpelExpression.getValue(EvaluationContext, Object, Class<T>) line: 102
MessagingMethodInvokerHelper<T>(AbstractExpressionEvaluator).evaluateExpression(Expression, Object, Class<T>) lin
MessagingMethodInvokerHelper<T>.processInternal(ParametersWrapper) line: 225
MessagingMethodInvokerHelper<T>.process(Message<?>) line: 125
MethodInvokingMessageProcessor<T>.processMessage(Message<?>) line: 73
MethodInvokingTransformer(AbstractMessageProcessingTransformer).transform(Message<?>) line: 56
MessageTransformingHandler.handleRequestMessage(Message<?>) line: 67
MessageTransformingHandler(AbstractReplyProducingMessageHandler).handleMessageInternal(Message<?>) line: 97
MessageTransformingHandler(AbstractMessageHandler).handleMessage(Message<?>) line: 73
SimpleMessageHandlerMetrics.handleMessage(Message<?>) line: 108
SimpleMessageHandlerMetrics.invoke(MethodInvocation) line: 88
ReflectiveMethodInvocation.proceed() line: 172
JdkDynamicAopProxy.invoke(Object, Method, Object[]) line: 202
$Proxy1.handleMessage(Message) line: not available
MessageHandlerChain$1.send(Message<?>) line: 154
MessagingTemplate.doSend(MessageChannel, Message<?>) line: 288
MessagingTemplate.send(MessageChannel, Message<P>) line: 149
MessageTransformingHandler(AbstractReplyProducingMessageHandler).sendMessage(Message<?>, Object) line: 175
MessageTransformingHandler(AbstractReplyProducingMessageHandler).sendReplyMessage(Message<?>, Object) line: 159
MessageTransformingHandler(AbstractReplyProducingMessageHandler).produceReply(Object, MessageHeaders) line: 124
MessageTransformingHandler(AbstractReplyProducingMessageHandler).handleResult(Object, MessageHeaders) line: 118
MessageTransformingHandler(AbstractReplyProducingMessageHandler).handleMessageInternal(Message<?>) line: 100
MessageTransformingHandler(AbstractMessageHandler).handleMessage(Message<?>) line: 73
SimpleMessageHandlerMetrics.handleMessage(Message<?>) line: 108
SimpleMessageHandlerMetrics.invoke(MethodInvocation) line: 88
ReflectiveMethodInvocation.proceed() line: 172
JdkDynamicAopProxy.invoke(Object, Method, Object[]) line: 202
$Proxy1.handleMessage(Message) line: not available
MessageHandlerChain.handleMessageInternal(Message<?>) line: 137
MessageHandlerChain(AbstractMessageHandler).handleMessage(Message<?>) line: 73
SimpleMessageHandlerMetrics.handleMessage(Message<?>) line: 108
SimpleMessageHandlerMetrics.invoke(MethodInvocation) line: 88
ReflectiveMethodInvocation.proceed() line: 172
JdkDynamicAopProxy.invoke(Object, Method, Object[]) line: 202
$Proxy3.handleMessage(Message) line: not available
UnicastingDispatcher.doDispatch(Message<?>) line: 115
UnicastingDispatcher.dispatch(Message<?>) line: 102
DirectChannel(AbstractSubscribableChannel).doSend(Message<?>, long) line: 77
DirectChannel(AbstractMessageChannel).send(Message<?>, long) line: 157
DirectChannel(AbstractMessageChannel).send(Message<?>) line: 128
NativeMethodAccessorImpl.invoke0(Method, Object, Object[]) line: not available [native method]
NativeMethodAccessorImpl.invoke(Object, Object[]) line: 39
DelegatingMethodAccessorImpl.invoke(Object, Object[]) line: 25
Method.invoke(Object, Object...) line: 597
AopUtils.invokeJoinpointUsingReflection(Object, Method, Object[]) line: 318
ReflectiveMethodInvocation.invokeJoinpoint() line: 183
ReflectiveMethodInvocation.proceed() line: 150
DirectChannelMetrics.monitorSend(MethodInvocation, MessageChannel, Message<?>) line: 113
DirectChannelMetrics.doInvoke(MethodInvocation, String, MessageChannel) line: 97
DirectChannelMetrics.invoke(MethodInvocation) line: 91
ReflectiveMethodInvocation.proceed() line: 172
JdkDynamicAopProxy.invoke(Object, Method, Object[]) line: 202
$Proxy0.send(Message) line: not available
ComponentInvocation.timeFlow(MessageChannel, PollableChannel) line: 62
ComponentInvocation.main(String[]) line: 48

SI Isn't Slow

- The pattern continues across other component types
- For a flow spanning 10 components, SI adds 10's of us
  - For most applications, domain logic dwarfs this
- If these us are critical we can save some
  - Requires wrapping our code with SI interfaces
  - Avoids introspection and reflection

Transactions Are

- Typically throughput drops by 2 orders of magnitude
- HA systems may need to handle duplicates anyway
  - Idempotent services make this easier
- If possible
  - Aim to process as much as possible
  - Prefer compensation to transactions

Tracking Latency

JMX

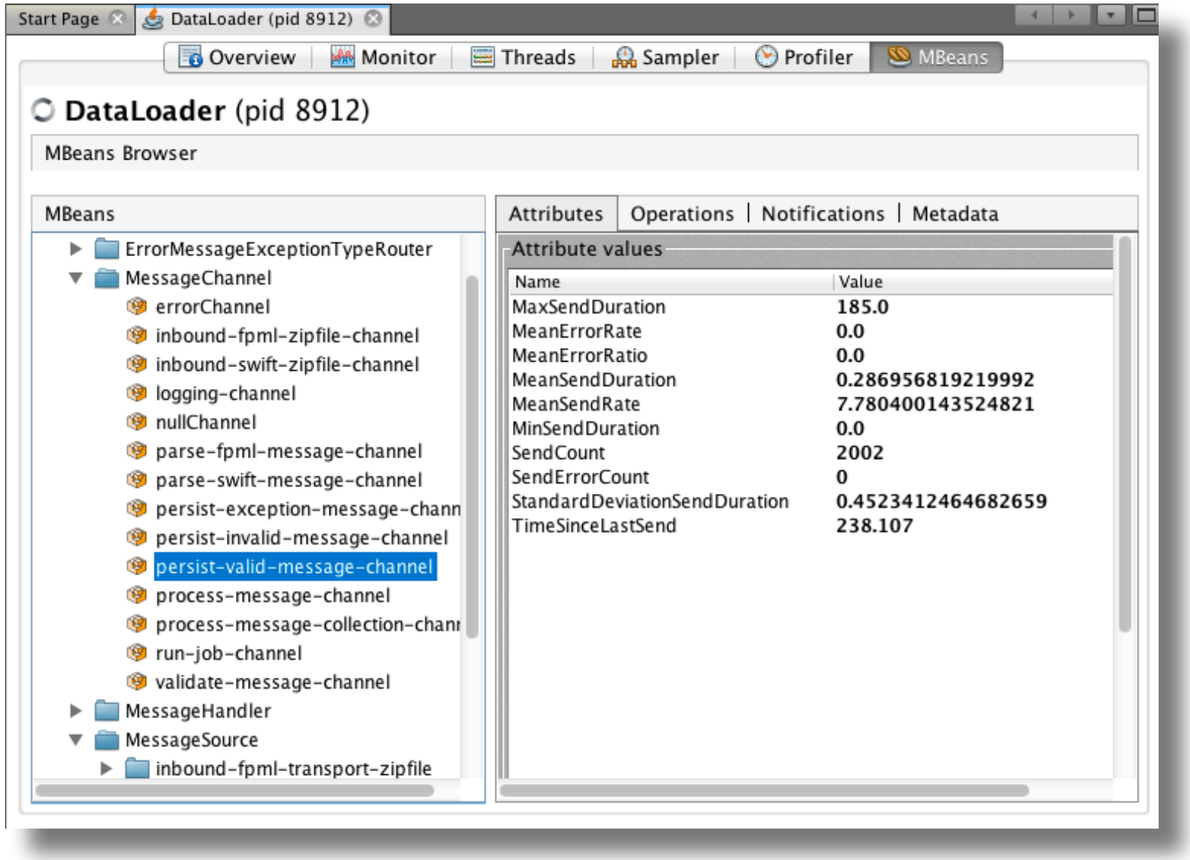
- Sometimes we need to track the processing time of messages in production
  - Depending on inbound adapter, outside timing might not be possible
- If we just need generalised statistics, SI's JMX support may be applicable

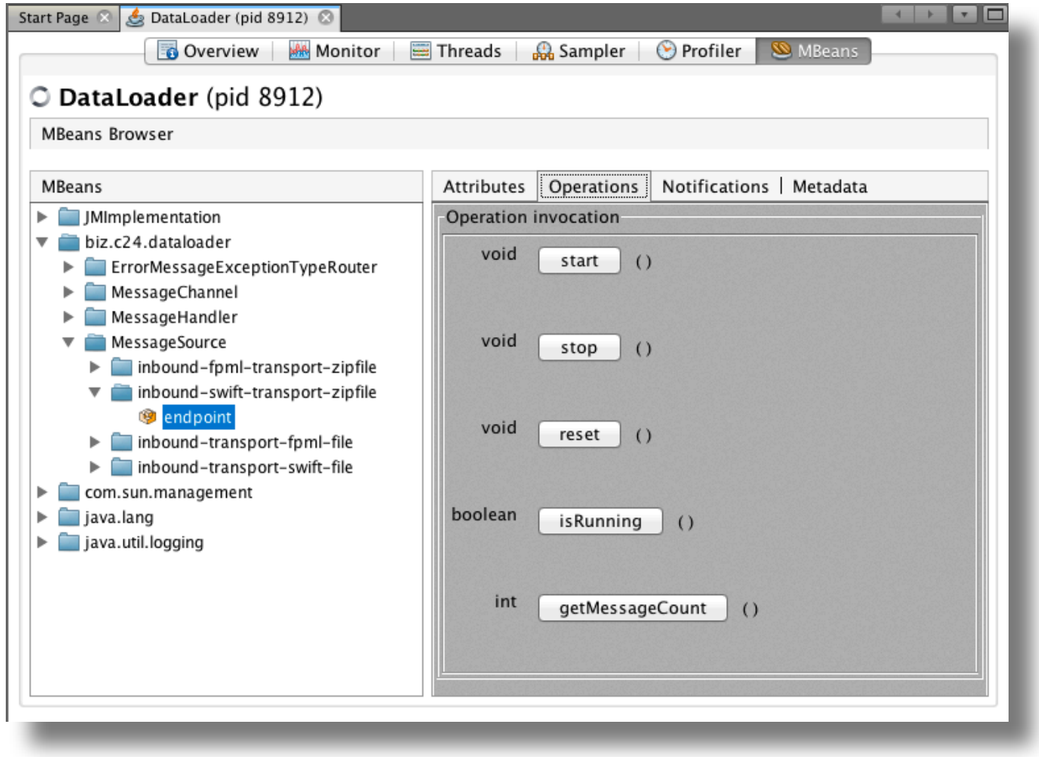
```
<jmx:mbean-export default-domain="biz.c24.springone" server="mbeanServerFactory"/>

<beans:bean id="mbeanServerFactory"
            class="org.springframework.jmx.support.MBeanServerFactoryBean">
    <beans:property name="locateExistingServerIfPossible" value="true"/>
</beans:bean>
```

JMX

- Every component proxied - adds ~3.5us to our previous test





Per-Message Stats

- We can track individual messages by stamping them at key points in the flow

- The simple solution

```
<header-enricher>
    <header name="startTime" expression="T(System).nanoTime()"></header>
</header-enricher>
...
<header-enricher>
    <header name="processingTime"
            expression="T(System).nanoTime() - headers[startTime]" />
</header-enricher>
```

- Can we beat it?

Result

- SpEL approach adds around 158us
- A custom transformer to set the headers adds around 10us

Keeping It Simple

Messages Like Monogamy, Threads Do Not

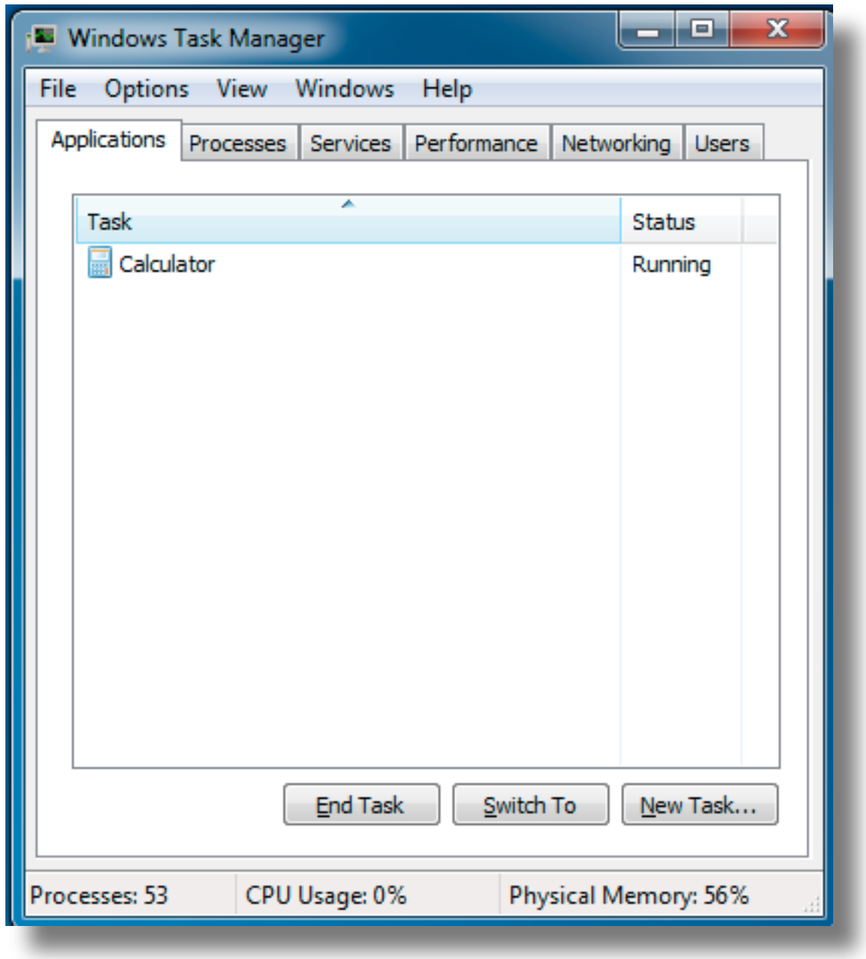
- Life is much simpler (& faster) if each message is processed by a single thread

- Handing messages between threads introduces complexity
  - Transactional barriers are broken
  - We have to deal with timeout
  - We have to handle unexpected replies

- There are always exceptions
  - Significant amounts of async IO or slow external services
  - Requirement to implement processing timeout

Windows Task Manager

- Not Responding == ?
- Running == ?



SI Gateway - Synchronous

- default-reply-timeout
  - Cannot abort long running process
  - Needed if your downstream processes can return null
    - Alternatively set requires-reply attribute on the null-generating component

SI Gateway - Asynchronous

- default-reply-timeout
  - Does what you'd expect
  - Remember your reply channel might still receive a response after this
    - Something needs to consume and process the message

Scaling Up

Multi-threading

- Assuming your components are re-entrant we can add more threads to the flow
- Some inbound adapters allow us to explicitly use multiple threads
  - e.g. concurrent-consumers on the inbound JMS adapter
- For others we need to create a thread pool and wire it into a channel

```
<task:executor id="receiptFileLoadingPool" pool-size="8" rejection-policy="CALLER_RUNS"/>

<channel id="inboundReceiptCsvFile">
    <dispatcher task-executor="receiptFileLoadingPool"/>
</channel>
```

Retail System

- Initial requirement is to parse receipt data from a queue

```
<receipt receiptId="bad337a6-940c-44f4-b5fd-44e0a9820867" customerId="20739" timestamp="2012-03-01T01:30:24">
   <item productId="239" quantity="1" price="2.93"/>
   <item productId="243" quantity="3" price="8.37"/>
   <item productId="244" quantity="3" price="8.97"/>
   <item productId="274" quantity="1" price="2.99"/>
</receipt>
```

Message Format

- Apart from inbound & outbound adapters, flow is simple:

```
<channel id="inbound.receipt.xml"/>

<!-- iO model of the object we're going to parse -->
<c24:model id="receiptXmlModel" base-element="biz.c24.retaildemo.model.xml.ReceiptElement"/>

<chain input-channel="inbound.receipt.xml" output-channel="persistReceipts">
    <int-c24:unmarshalling-transformer model-ref="receiptXmlModel"/>
    <!--  Transform to our normalised format -->
    <int-c24:transformer
            transform-class="biz.c24.retaildemo.transform.ReceiptXmlToJavaTransform”
            target-class="biz.c24.retaildemo.model.Receipt"/>
</chain>
```

- Executes in approx 80us - well within our SLA
  - Can go a lot faster but it makes the numbers easier for today

Acquisition

- Messages came in via an AMQP queue, but we'll use the file system for now

- We can scale up by adding a task-dispatcher to the channel
  - Tuning the max-messages-per-poll and poll-frequency is sensitive
- ...adding a task-dispatcher to the poller
- ...or clustering horizontally

- Life is good

Batch Data

- Some data comes in a batch file
- Parsing is a serial process
  - Processing an individual file becomes a single-threaded job again
  - 100,000 receipts @ 80us per receipt = 80s
- Instead we rapidly split the file and process the individual receipts in parallel

```
<chain input-channel="inbound.receipt.multixml" output-channel="inbound.receipt.xml">
    <splitter>
        <beans:bean class="biz.c24.retaildemo.si.StructuredFileSplitter">
            <beans:constructor-arg value="&lt;receipt.*"/>
        </beans:bean>
    </splitter>
</chain>
```

Business Is Good

- Which means more transactions
- Which means the file grows...

```
<receipt receiptId="f7ade2e7-3be4-4155-998a-973a3c2684e4" customerId="45276"
timestamp="2012-03-01T01:09:41">
        <item productId="296" quantity="2" price="4.98"/>
        <item productId="299" quantity="5" price="14.95"/>
        <item productId="279" quantity="3" price="14.97"/>
        <item productId="30" quantity="2" price="11.98"/>
        <item productId="401" quantity="1" price="8.99"/>
        <item productId="425" quantity="2" price="17.32"/>
        <item productId="409" quantity="1" price="8.99"/>
        <item productId="407" quantity="1" price="8.99"/>
        <item productId="415" quantity="1" price="8.99"/>
        <item productId="165" quantity="1" price="2.29"/>
</receipt>
<receipt receiptId="6a9d41dc-e8b0-44e9-800b-01a3add8474d" customerId="65803"
timestamp="2012-03-01T01:05:10">
        <item productId="2" quantity="12" price="27.48"/>
        <item productId="26" quantity="3" price="6.75"/>
        <item productId="203" quantity="3" price="19.5"/>
</receipt>
<receipt receiptId="100cdec2-d647-45bc-bd2e-d623032d340a" customerId="67856" timestamp="2012-03-01T01:43:04">
        <item productId="242" quantity="10" price="54.0"/>
        <item productId="290" quantity="10" price="17.90"/>
        <item productId="314" quantity="18" price="33.12"/>
        <item productId="347" quantity="3" price="5.55"/>
        <item productId="345" quantity="4" price="7.40"/>
        <item productId="337" quantity="3" price="8.07"/>
        <item productId="351" quantity="2" price="4"/>
        <item productId="331" quantity="2" price="6.98"/>
        <item productId="335" quantity="1" price="1.79"/>
        <item productId="344" quantity="1" price="1.85"/>
        <item productId="328" quantity="4" price="8.76"/>
        <item productId="319" quantity="1" price="1.25"/>
        <item productId="154" quantity="1" price="2.13"/>
        <item productId="416" quantity="3" price="17.97"/>
        <item productId="444" quantity="1" price="6.39"/>
        <item productId="166" quantity="2" price="5.98"/>
</receipt>
<receipt receiptId="ba9a6cd2-4dab-4957-b48d-418e1be06f10" customerId="68212" timestamp="2012-03-01T01:44:16">
        <item productId="307" quantity="56" price="196.0"/>
        <item productId="348" quantity="24" price="96"/>
        <item productId="353" quantity="18" price="71.82"/>
        <item productId="414" quantity="5" price="129.40"/>
        <item productId="163" quantity="4" price="40.8"/>
</receipt>
<receipt receiptId="3970b1e4-83f0-404f-a974-1bfaac0a02e8" customerId="72204" timestamp="2012-03-01T01:05:09">
        <item productId="293" quantity="14" price="62.30"/>
        <item productId="354" quantity="13" price="37.31"/>
        <item productId="333" quantity="8" price="15.92"/>
        <item productId="174" quantity="2" price="9.98"/>
</receipt>
```

Splitters

- Splitters generate a single collection with everything in it
  - Good for small inputs
  - Bad for large inputs

- If we use a Service Activator we can push individual messages on the channel

What Went Wrong?

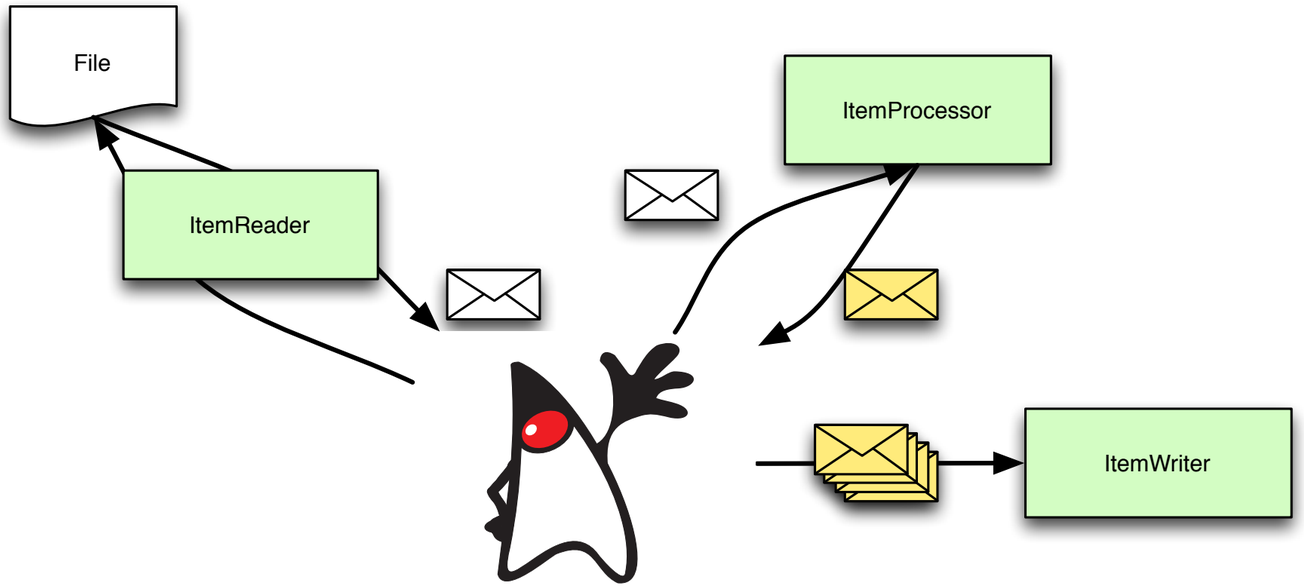
- Our splitting process works well, but by design it's faster than the downstream processing
- We could create a fixed capacity, blocking queue to mediate
  - How long does the thread wait when offering to the queue before it aborts?
  - What rate do we throttle the splitter at?
- Without this we create a bottleneck which at some point exhausts the memory
  - With it we risk creating an application that hangs indefinitely
- Remember our original maxim - try and process a message in a single thread

Spring Batch

- Unsurprisingly, it's well suited to processing large batch files

- Define a Job as a set of Steps
  - Flow between Steps depends on outcome
  - Within a Step records can be processed in Chunks

- Today we'll use a single Step



Solving the Splitting Problem

- Use the same thread to split out the message and to process it
  - Effectively we're streaming the file hence auto-throttling
- We have (minimal) contention on the file read
  - Make the splitting process fast

The Job

```
<!-- Declare an ItemReader -->
<bat-c24:item-reader id="ioItemReader" model-ref="receiptModel" elementStartPattern="&lt;receipt.*">
     <bat-c24:file-source encoding="UTF-8"/>
 </bat-c24:item-reader>


<!-- Use an ItemProcessor to transform source model format to our internal format -->
<bat-c24:transform-item-processor id="ioItemProcessor" transform-ref="receiptXmlToReceiptTransform"
                                  target-class="biz.c24.retaildemo.model.Receipt"/>



 <job id="fileLoader">
    <step id="parseFile">
       <tasklet task-executor="receiptFileLoadingPool" throttle-limit="4">
          <chunk reader="ioItemReader" processor="ioItemProcessor" writer="monitoringItemWriter"
                 commit-interval="100" processor-transactional="false"/>
       </tasklet>
    </step>
 </job>
```

Scratching the Surface

- We can do lots of other cool things with Spring Batch
  - Scheduled Jobs
  - Proper tracking, retry, restart etc
  - Skip exceptions, set skip limits...

But I Like Spring Integration

- So does Spring Batch!
- We can launch a Spring Batch job from Spring Integration
- Create an ItemWriter to push messages back into Spring Integration
- Push exceptions into Spring Integration flow for handling

Spring Batch Integration

- First we need to convert the payload to a JobLaunchRequest

```
<!-- Creates a job launch request from a File payload -->
<beans:bean class="org.springframework.batch.admin.integration.FileToJobLaunchRequestAdapter">
   <beans:property name="job" ref="fileLoader" />
</beans:bean>
```

- Then we need something to trigger the Spring Batch job launcher

```
<!-- Launches a Spring Batch job from a JobLaunchRequest payload -->
<beans:bean id="jobMessageLauncher"
            class="org.springframework.batch.integration.launch.JobLaunchingMessageHandler">
   <beans:constructor-arg ref="jobLauncher"/>
</beans:bean>
```

Stitch Them Together

```
<!-- Launch a Spring Batch job from our File payload -->
<int:chain input-channel="run-job-channel" output-channel="errorChannel">
   <!-- Create a job launch request -->
   <int:service-activator>
      <beans:bean class="org.springframework.batch.admin.integration.FileToJobLaunchRequestAdapter">
            <beans:property name="job" ref="fileLoader" />
      </beans:bean>
    </int:service-activator>

    <!-- Launch the job -->
    <int:service-activator ref="jobMessageLauncher"/>

    <!-- Because we're using a synchronous job launch, the JobExecution will contain the final status  -->
    <!-- Get any exceptions that the job generated -->
    <int:transformer expression="payload.allFailureExceptions"/>
    <int:splitter/>
</int:chain>
```

- Ensure we handle any exceptions that cause the job to fail

Pushing Items Back Into SI

- Use a gateway to proxy the ItemWriter interface

```
<!-- Provide an ItemWriter that creates SI messages for successful CDOs -->
<gateway service-interface="org.springframework.batch.item.ItemWriter"
         id="cdoChannelWriter"
         default-request-channel="process-message-collection-channel" />
```

- Gateway is synchronous
  - We'll continue to use the same thread to process them in SI

```
<!-- Split the chunks from the cdoChannelWriter has given us into individual messages -->
<int:splitter input-channel="process-message-collection-channel"
              output-channel="process-message-channel"/>
```

Skipping Exceptions

- We can designate certain failures as non-fatal

```
<chunk ... skip-limit="10">

    <!-- On parsing or validation exceptions, continue processing the rest of the file if possible -->
    <skippable-exception-classes>
        <include class="org.springframework.batch.item.ParseException"/>
        <include class="org.springframework.batch.item.validator.ValidationException"/>
    </skippable-exception-classes>

     <listeners>
     <!-- Make sure that skipped exceptions are pushed back into our core error handling flow -->
         <listener ref="exceptionChannelWriter"></listener>
     </listeners>
</chunk>
```

- Normally parse failures would be fatal
  - Where does the next record start?
  - Splitting approach allows us to process the rest of the file

Same Approach for Skipped Exceptions

```
<!-- Provide an ItemWriter that creates SI messages from Exceptions -->
<int:gateway id="exceptionChannelWriter"
             service-interface="org.springframework.batch.core.SkipListener"
             default-request-channel="errorChannel"/>
```

Processing Zip Files

- Traditional approach is to pre-process
- This creates something else to schedule, monitor, diagnose...
- We can simply change the ItemReader to something that can process zip files

```
<!-- Declare an ItemReader -->
<bat-c24:item-reader id="ioItemReader" model-ref="receiptModel" elementStartPattern="&lt;receipt.*">
     <bat-c24:zip-file-source encoding="UTF-8"/>
 </bat-c24:item-reader>
```

Thank You!

The source code demonstrated during this presentation can be downloaded from
https://github.com/C24-Technologies/c24-sample-springperformance