Stream caching

Stream caching

While stream types (like StreamSource, InputStream and Reader) are commonly used in messaging for performance reasons, they also have an important drawback: they can only be read once. In order to be able to work with message content multiple times, the stream needs to be cached.

Streams are caching in memory. In Camel 2.0, large stream messages (over 64 Kb in Camel 2.11 or older, and 128 kb from Camel 2.12 onwards) will be cached in a temporary file instead – Camel itself will handle deleting the temporary file once the cached stream is no longer necessary.

In Camel 2.0 stream cache is default **disabled** out of the box.
In Camel 1.x stream cache is default **enabled** out of the box.

StreamCache Affects your payload object

The StreamCache will affect your payload object as it will replace the Stream payload with a `org.apache.camel.StreamCache` object. This `StreamCache` is capable of being re-readable and thus possible to better be routed within Camel using redelivery or Content Based Router or the likes.

However to not change the payload under the covers without the end user really knowing we changed the default in Camel 2.0 to **disabled**. So in Camel 2.0 you have to explicit enable it if you want to use it.

If using Camel 2.12 onwards then see about StreamCachingStrategy further below which is the recommended way to configure stream caching options.

Enabling stream caching

In Apache Camel, you can explicitly enable stream caching for a single route with the **`streamCaching`** DSL method:

```
from("jbi:service:http://foo.bar.org/MyService")
    .streamCaching()
    .to("jbi:service:http://foo.bar.org/MyOtherService");
```

In Spring XML you enable it by setting the `streamCache="true"` attribute on the `route` tag.

```
<route streamCache="true">
   <from uri="jbi:service:http://foo.bar.org/MyService"/>
   <to uri="jbi:service:http://foo.bar.org/MyOtherService"/>
</route>
```

Scopes

StreamCache supports the global and per route scope. So by setting the streamCache attribute on camelContext you can enable/disable it globally.

```
<camelContext streamCache="true">
   ...
</camelContext>
```

The route scope is configured by the `streamCache` attribute on the `<route>` tag such as:

```
<route streamCache="true">
   <from uri="jbi:service:http://foo.bar.org/MyService"/>
   <to uri="jbi:service:http://foo.bar.org/MyOtherService"/>
</route>
```

You can mix and match for instance you can enable it globally and disable it on a particular route such as:

```
<camelContext streamCache="true">
  <route>
    <from uri="jbi:service:http://foo.bar.org/MyService"/>
    <to uri="jbi:service:http://foo.bar.org/MyOtherService"/>
  </route>

  <route streamCache="false">
    <from uri="jms:queue:foo"/>
    <to uri="jms:queue:bar"/>
  </route>

</camelContext>
```

Enabling from Java DSL

You can enable stream cache by setting the property on CamelContext, for instance in a RouteBuilder class:

```
  context.setStreamCache(true);

  from("jbi:service:http://foo.bar.org/MyService")
     .to("jbi:service:http://foo.bar.org/MyOtherService");
```

Disable stream caching explicitly

If you have enabled stream caching globally on the CamelContext and you want to disable it for certain routes in a selective manner, you can use the following syntax.

Spring DSL:

```
<camelContext streamCache="true">
  <route streamCache="false">
    <from uri="jetty:http://0.0.0.0:9090"/>
    <to uri="file:target/incoming"/>
  </route>
</camelContext>
```

Java DSL:

```
  context.setStreamCache(true);

  from("jetty:http://0.0.0.0:9090").noStreamCaching()
     .to("file:target/incoming");
```

Streaming cache to files

When stream cache is enabled it will by default spool big streams to files instead of keeping them in memory. The default threshold is 64kb but you can configure it with the following properties:

Property	Default	Description
CamelCachedOutputStreamBufferSize	2kb	**Camel 2.9.4, 2.10.2, 2.11.0:** Size in bytes of the buffer used in the stream.
CamelCachedOutputStreamThreshold	64kb or 128kb	64kb for **Camel 2.11** or older. 128kb for **Camel 2.12** onwards. Size in bytes when the stream should be spooled to disk instead of keeping in memory. Use a value of 0 or negative to disable it all together so streams is always kept in memory regardless of their size.
CamelCachedOutputStreamOutputDirectory	java.io.tmpdir	Base directory where temporary files for spooled streams should be stored.

CamelCachedOutputStreamCipherTransformation	null	**Camel 2.11.0:** If set, the temporary files are encrypted using the specified cipher transformation (i.e., a valid stream or 8-bit cipher name such as "RC4", "AES/CTR/NoPadding". An empty name "" is treated as null).

You set these properties on the CamelContext as shown below, where we use a 1mb threshold to spool to disk for messages bigger than 1mb:

```
context.getProperties().put(CachedOutputStream.TEMP_DIR, "/tmp/cachedir");
context.getProperties().put(CachedOutputStream.THRESHOLD, "1048576");
context.getProperties().put(CachedOutputStream.BUFFER_SIZE, "131072");
// to enable encryption using RC4
// context.getProperties().put(CachedOutputStream.CIPHER_TRANSFORMATION, "RC4");
```

And in XML you do

```
<camelContext xmlns="http://camel.apache.org/schema/blueprint">

  <!-- disable stream caching spool to disk -->
  <properties>
    <property key="CamelCachedOutputStreamOutputDirectory" value="/tmp/cachedir"/>
    <property key="CamelCachedOutputStreamThreshold" value="1048576"/>
    <property key="CamelCachedOutputStreamBufferSize" value="131072"/>
  </properties>
```

Disabling spooling to disk

You can disable spooling to disk by setting a threshold of 0 or a negative value.

```
// disable spooling to disk
context.getProperties().put(CachedOutputStream.THRESHOLD, "-1");
```

And in XML you do

```
<camelContext xmlns="http://camel.apache.org/schema/blueprint">

  <!-- disable stream caching spool to disk -->
  <properties>
    <property key="CamelCachedOutputStreamThreshold" value="-1"/>
  </properties>
```

Using StreamCachingStrategy

Available as of Camel 2.12

Stream caching is from Camel 2.12 onwards intended to be configured using `org.apache.camel.spi.StreamCachingStrategy`.
The old kind of configuration using properties on the CamelContext has been marked as deprecated.

The strategy has the following options:

Option	Default	Description
`spoolDirectory`	`${java.io.tmpdir}/camel/camel-tmp-#uuid#`	Base directory where temporary files for spooled streams should be stored. This option supports naming patterns as documented below.
`spoolChiper`	`null`	If set, the temporary files are encrypted using the specified cipher transformation (i.e., a valid stream or 8-bit cipher name such as "RC4", "AES/CTR/NoPadding". An empty name "" is treated as null).
`spoolThreshold`	`128kb`	Size in bytes when the stream should be spooled to disk instead of keeping in memory. Use a value of 0 or negative to disable it all together so streams is always kept in memory regardless of their size.
`spoolUsedHeapMemoryThreshold`	`0`	A percentage (1 to 99) of current used heap memory to use as threshold for spooling streams to disk. The upper bounds is based on heap committed (guaranteed memory the JVM can claim). This can be used to spool to disk when running low on memory.

`spoolUsedHeapMemoryLimit`	`Max`	If `spoolUsedHeapMemoryThreshold` is in use, then whether the used heap memory upper limit is either `Max` or `Committed`.
`anySpoolRules`	`false`	Whether any or all `SpoolRule}}s must return {{true` to determine if the stream should be spooled or not. This can be used as applying AND/OR binary logic to all the rules. By default its AND based.
`bufferSize`	`4096`	Initial size if in-memory created stream buffers.
`removeSpoolDirectoryWhenStopping`	`true`	Whether to remove the spool directory when stopping CamelContext.
`statisticsEnabled`	`false`	Whether utilization statistics is enabled. By enabling this you can see these statics for example with JMX.

SpoolDirectory naming pattern

The following patterns is supported:

- #uuid# = a random UUID
- #camelId# = the CamelContext id (eg the name)
- #name# - same as #camelId#
- #counter# - an incrementing counter
- #bundleId# - the OSGi bundle id (only for OSGi environments)
- #symbolicName# - the OSGi symbolic name (only for OSGi environments)
- #version# - the OSGi bundle version (only for OSGi environments)
- ${env:key} - the environment variable with the key
- ${key} - the JVM system property with the key

A could of examples, to store in the java temp directory with a sub directory using the CamelContext name:

```
context.getStreamCachingStrategy().setSpoolDirectory"${java.io.tmpdir}#name#/");
```

To store in KARAF_HOME/tmp/bundleId directory

```
context.getStreamCachingStrategy().setSpoolDirectory"${env:KARAF_HOME}/tmp/bundle#bundleId#");
```

Using StreamCachingStrategy in Java

You can configure the `StreamCachingStrategy` in Java as shown below:

```
context.getStreamCachingStrategy().setSpoolDirectory"/tmp/cachedir");
context.getStreamCachingStrategy().setSpoolThreshold(64 * 1024);
context.getStreamCachingStrategy().setBufferSize(16 * 1024);
// to enable encryption using RC4
// context.getStreamCachingStrategy().setSpoolChiper("RC4");
```

And remember to enable Stream caching on the CamelContext or on routes

```
context.setStreamCaching(true);
```

Using StreamCachingStrategy in XML

And in XML you do:

```
  <camelContext streamCache="true" xmlns="http://camel.apache.org/schema/blueprint">

    <streamCaching id="myCacheConfig" bufferSize="16384" spoolDirectory="/tmp/cachedir" spoolThreshold="65536"/>

    <route>
      <from uri="direct:c"/>
      <to uri="mock:c"/>
    </route>

  </camelContext>
```

You can also define a <bean> instead of using the <streamCaching> tag:

And in XML you do

```
<!-- define a bean of type StreamCachingStrategy which CamelContext will automatic use -->
<bean id="streamStrategy" class="org.apache.camel.impl.DefaultStreamCachingStrategy">
  <property name="spoolDirectory" value="/tmp/cachedir"/>
  <property name="spoolThreshold" value="65536"/>
  <property name="bufferSize" value="16384"/>
</bean>

<!-- remember to enable stream caching -->
<camelContext streamCaching="true" xmlns="http://camel.apache.org/schema/spring">
```

Using spoolUsedHeapMemoryThreshold

By default stream caching will spool only big payloads (128kb or bigger) to disk. However you can also set the spoolUsedHeapMemoryThreshold option which is a percentage of used heap memory. This can be used to also spool to disk when running low on memory.

For example with:

```
    <streamCaching id="myCacheConfig" spoolDirectory="/tmp/cachedir" spoolUsedHeapMemoryThreshold="70"/>
```

Then notice that as spoolThreshold is default enabled with 128kb, then we have both thresholds in use (spoolThreshold and spoolUsedHeapMemoryThreshold). And in this example then we only spool to disk if payload is > 128kb and that used heap memory is > 70%. The reason is that we have the option `anySpoolRules` as default `false`. That means both rules must be `true` (eg AND).

If we want to spool to disk if either of the rules (eg OR), then we can do:

```
    <streamCaching id="myCacheConfig" spoolDirectory="/tmp/cachedir" spoolUsedHeapMemoryThreshold="70"
anySpoolRules="true"/>
```

If we only want to spool to disk if we run low on memory then we can set:

```
    <streamCaching id="myCacheConfig" spoolDirectory="/tmp/cachedir" spoolThreshold="-1"
spoolUsedHeapMemoryThreshold="70"/>
```

... then we do not use the spoolThreshold rule, and only the heap memory based is in use.

By default the upper limit of the used heap memory is based on the maximum heap size. Though you can also configure to use the committed heap size as the upper limit, this is done using the `spoolUsedHeapMemoryThreshold` option as shown below:

```
    <streamCaching id="myCacheConfig" spoolDirectory="/tmp/cachedir" spoolUsedHeapMemoryThreshold="70"
spoolUsedHeapMemoryLimit="Committed"/>
```

Using custom SpoolRule implementations

You can implement your custom rules to determine if the stream should be spooled to disk. This can be done by implementing the interface `org.apache.camel.spi.StreamCachingStrategy.SpoolRule` which has a single method:

```
boolean shouldSpoolCache(long length);
```

The length is the length of the stream.

To use the rule then add it to the `StreamCachingStrategy` as shown below:

```
SpoolRule mySpoolRule = ...
context.getStreamCachingStrategy().addSpoolRule(mySpoolRule);
```

And from XML you need to define a <bean> with your custom rule

```
<bean id="mySpoolRule" class="com.foo.MySpoolRule"/>

<streamCaching id="myCacheConfig" spoolDirectory="/tmp/cachedir" spoolRules="mySpoolRule"/>
```

Using the `spoolRules` attribute on <streamCaching>. if you have more rules, then separate them by comma.

```
<streamCaching id="myCacheConfig" spoolDirectory="/tmp/cachedir" spoolRules="mySpoolRule,myOtherSpoolRule"/>
```

How it works?

In order to determine if a type requires caching, we leverage the type converter feature. Any type that requires stream caching can be converted into an **`org.apache.camel.StreamCache`** instance.