Advanced configuration of CamelContext using Spring

Advanced configuration of CamelContext using Spring

When using Spring the CamelContext can be pre configured based on defined beans in spring XML.
This wiki page documentes these features. Most of these features requires **Camel 2.0**.

What can be configured

The following functions can be configured:

- Class resolvers
- Lifecycle factories
- Registry for lookup
- Debugger, Tracer, Delay and UuidGenerator
- Intercept
- Graceful Shutdown
- Stream caching
- Logging

Camel will configure these functions by doing a lookup in the Spring bean registry to find beans of the given type
When Camel finds and uses any of these it logs at `INFO` level.

The following list all requires at most 1 beans defined. If there are more than 1 bean of this type, then Camel will **not** use it.

Type	Number of beans	Description
PackageScanClassResolver	0..1	To use a 3rd party package scan resolver. More details at Pluggable Class Resolvers.
ClassResolver	0..1	To use a 3rd party class resolver. More details at Pluggable Class Resolvers.
FactoryFinderResolver	0..1	To use a 3rd party factory finder.
Registry	0..1	To use a 3rd party bean registry. By default Camel will use Spring ApplicationContext as registry.
Debugger	0..1	To use a Debugger usually for tooling.
Tracer	0..1	To use a 3rd party Tracer.
TraceFormatter	0..1	To use a bean that has the tracing options configured.
HandleFault	0..1	To use a 3rd part fault handler to handle FAULT messages.
Delayer	0..1	To use a 3rd part Delayer.
ManagementStrategy	0..1	**Camel 2.1:** To use a 3rd part strategy for management, for example JMX management.
ManagementNamingStrategy	0..1	**Camel 2.6:** To use a 3rd part strategy for naming MBeans for management.
NodeIdFactory	0..1	**Camel 2.10:** To use a 3rd part node id factory.
EventFactory	0..1	**Camel 2.1:** To use a 3rd part event factory.
EventNotifier	0..1	**Camel 2.1:** To use a 3rd part event notifier. In Camel 2.2 onwards you can have multiple notifiers, see next table.
InflightRepository	0..1	**Camel 2.1:** To use a 3rd part inflight repository.
ShutdownStrategy	0..1	**Camel 2.2:** To use a 3rd part shutdown strategy.
ExecutorServiceStrategy	0..1	**Camel 2.3 - 2.8.x:** To use a 3rd part executor service strategy. More details at Threading Model.
ExecutorServiceManager	0..1	**Camel 2.9:** To use a 3rd part executor service manager. More details at Threading Model.
ThreadPoolFactory	0..1	**Camel 2.9:** To use a 3rd part thread pool factory. More details at Threading Model.
ProcessorFactory	0..1	**Camel 2.4:** To use a 3rd part processor factory.
UuidGenerator	0..1	**Camel 2.5:** To use a 3rd part UuidGenerator.
StreamCachingStrategy	0..1	**Camel 2.12:** To use a 3rd part Stream caching strategy.
UnitOfWorkFactory	0..1	**Camel 2.12.3/2.13:** To use 3rd part `UnitOfWork` implementations created by the factory.

RuntimeEndpointRegistry	0..1	**Camel 2.13.1:** To use a 3rd party RuntimeEndpointRegistry implementation.
Logger	0..1	**Camel 2.12.4/2.13.1:** To use provided org.slf4j.Logger for Log component and log() EIP.
AsyncProcessorAwaitManager	0..1	**Camel 2.15:** To use a 3rd part async process await manager.
ModelJAXBContextFactory	0..1	**Camel 2.15.2:** To use a 3rd party model JAXB ContextFactory
MessageHistoryFactory	0..1	**Camel 2.17:** To use a 3rd party MessageHistoryFactory implementation.

And the following options have support for any number of beans defined.

Type	Number of beans	Description
InterceptStrategy	0..n	To use your own Intercept that intercepts every processing steps in all routes in the CamelContext. For instance you can use this to do an AOP like performance timer interceptor.
LifecycleStrategy	0..n	**Camel 2.1:** To use 3rd party lifecycle strategies. By default Camel uses a JMX aware that does JMX instrumentation.
EventNotifier	0..n	**Camel 2.2:** To use 3rd part event notifiers.
RoutePolicyFactory	0..n	**Camel 2.14:** To use a 3rd party route policy factory to create a route policy for every route.

Camel will log at `INFO` level if it pickup and uses a custom bean using `org.apache.camel.spring.CamelContextFactoryBean` as name.

Using container wide interceptors

Imagine that you have multiple CamelContext and you want to configure that they all use the same container wide interceptor. How do we do that? Well we can leverage the fact that Camel can auto detect and use custom interceptors. So what we simply do is to define our interceptor in the spring xml file. The sample below does this and also define 2 camel contexts. The sample is based on unit test.

Error formatting macro: snippet: java.lang.NullPointerException

Okay lets build our interceptor to simply count the number of interceptions. This is quite easy as we can just implement this logic in our implementation directly as the code below illustrates:

Error formatting macro: snippet: java.lang.NullPointerException

When Camel boots up it logs at `INFO` level the container wide interceptors it have found:

```
INFO  CamelContextFactoryBean        - Using custom intercept strategy with id: myInterceptor and
implementation:org.apache.camel.spring.interceptor.ContainerWideInterceptor@b84c44
```

Notice: If we have more than 1 container wide interceptor, we can just define them as spring bean. Camel will find and use them.

See Also

- Spring
- Spring JMS Tutorial
- Creating a new Spring based Camel Route
- Spring example
- Xml Reference