Camel 2.3 - ThreadPool Configuration

Design Notes for ThreadPool Configuration

CAMEL-1588 is the ticket for a new and improved thread pool configuration for Apache Camel. Its intended for Camel 2.3.

Scope

Camel uses thread pool in various places such as EIP patterns, Components, Async API and whatnot. The aim is to improved and allow easier to configure those thread pools in a more streamlined manner. The goal is to offer both a fine grained configuration where you can tweak individual pools and have more coarse grained configuration with fallback to *global* settings etc.

Outside scope

Some components provides their own thread pool configuration and management which Camel of course cannot and should not try to tailor with. For example Jetty is such an example.

Usages of thread pools in Camel

Currently Camel uses thread pools in **camel-core** in the following areas:

- `DefaultComponent` - Optional thread pool for components in need of such
- `DefaultEndpoint` - Optional thread pool for endpoints in need of such
- `DefaultProducerTemplate` - Used by the Async API of this template
- `ScheduledPollConsumer` - Needs a `ScheduledExecutorService` to schedule its tasks
- `RecipientListDefinition` - The Recipient List EIP pattern
- `SplitDefinition` - The Splitter EIP pattern
- `ThreadsDefinition` - The `threads` DSL
- `ToDefinition` - Used by the `toAsync` variation
- `WireTapDefinition` - The Wire Tap EIP pattern
- `MulticastProcessor` - The underlying thread pool
- `OnCompletionProcessor` - For sending async on completion routes
- `SendAsyncProcessor` - The `toAsync` processor variation
- `ThreadsProcessor` - The underlying thread pool
- `WireTapProcessor` - For sending async wire taps
- `SedaConsumer` - To support the `concurrentConsumers` and `multipleConsumers` options (uses separate pools)

Existing configuration

You can configure the thread pool using the **setExecutorService** setter methods that usually exists on those places where its in use. Some EIP patterns offer a `executorServiceRef` option to refer to some pool to be looked up in the Registry.

We should ensure all EIPs can be configured to use a custom thread pool in a nice and easy way. **DONE**

Using default ThreadPools

We should use `CachedThreadPool` from the JDK Core as its the best general purpose pool for many short lived tasks, which is what Camel really does. Processing many messages in a short live. **DONE**

Only used SingleExecutorService for background tasks, and ScheduledExecutorService for scheduled tasks **DONE**

ThreadPool scope

It should be possible to configure a thread pool on either per CamelContext level or per Route level, such as you can do with AutoStartup and the likes. Then you can say, eg this route should use this pool, which have 20/50 in the pool size etc. **CHANGE OF PLAN**

Thread pool configuration by rules

It should be possible to to define a set of rules which matches which thread pool a given source should use.
It should be pattern based so you can say all EIPs should use this pool, all endpoints that pool etc.

A ruleset something like this:

```
<threadPoolRule route="*" source="Aggregator" executorServiceRef="myAggPool"/>
<threadPoolRule route="*" source="To" executorServiceRef="mySendPool"/>
<threadPoolRule route="route3" source="*" executorServiceRef="myRoute3Pool"/>
```

Where it will match against route first, so if we got a route3 then it will pick among those
It should be possible to use wildcard and reg exp in the `route` and `source` attributes.

Status: Consider for the future

Default thread pool profile

It should be possible to set a default `ThreadPoolProfile` which Camel will use when creating thread pools for the EIPs and whatnot. Then you can set default settings and have that leveraged out of the box.

```
<threadPoolProfile id="myDefaultProfile"
                           defaultProfile="true"
                           poolSize="5" keepAliveTime="25" maxPoolSize="15" maxQueueSize="250" rejectedPolicy="
Abort"/>
```

If none defined, then Camel should use a sensible default of

- poolSize = 10
- maxPoolSize = 20
- keepAliveTime = 60 seconds
- maxQueueSize = 1000
- rejectedPolicy = CallerRuns

And it should validate that only **one** `defaultProfile=true` can be set.

Status: **DONE**

The problem with shutdown and restarting pools

The ExecutorService API does not allow to restart a thread pool, which is PITA. So we need to find a better strategy for stopping vs. shutdown.
Currently when we stop we also terminate the threadpool, and then re-create it on start. This only works for default pools which we can create again.
But for custom thread pools we have no way to create them again as the pool is already created when its given to us.

We may have to only shutdown thread pools if CamelContext is stopping. And then if end user stop a route from JMX we can keep the thread pool around.
Only issue is the scheduled pool should stop scheduling tasks, which may be a bit more trickier to avoid.

We have introduced a `ShutdownableService` to expose a `shutdown` method which services can implement for their shutdown logic.
Then we can only shutdown thread pools in `doShutdown()` and not as before in `doStop()`.

By letting Camel keep track of created thread pool, then Camel knows which pools shutdown when its stopping. Then the need for `doShutdown` is not as apparent as before, but its good to have this state in the lifecycles as well, for future needs.

Status: **DONE**

The problem with Component, Endpoint

The DefaultComponent and DefaultEndpoint exposes API to get an ExecutorService. We should remove these API as you should use `ExecutorService Strategy` from `CamelContext` to obtain a thread pool. **DONE**

Managed thread pool

Check whether the thread pools is managed by default and avail in JConsole. If not we should probably at least expose some read-only data about the pools. **DONE**

Spring Factory for creating custom pools

Create a Spring XML DSL for defining thread pools using custom options such as corePoolSize, maxPoolSize, keepAlive, thread name etc. **DONE**

Pluggable ExecutorService SPI

We need a `org.apache.camel.spi.ExeuctorServieStrategy` which is pluggable so end users can plugin their own strategy how to create thread pools. They can leverage a WorkManager API from J2EE server etc. **DONE**

Customizable thread name

We should offer a simple pattern syntax so end users can customize the pattern the thread name is created with: eg Something like: `Camel Thread ${counter} - ${name}`. Where counter and suffix is dynamic parameters. The counter is an unique incrementing thread counter. And name is provided from the source which as a way to name thread, such as a seda endpoint uri. **DONE**

EIP should mandate an ExecutorService

If the EIPS which leverages a ExecutorService, mandates its being created and passed to it, we can enforce creating/lookup the pool during route creation, which allows us to have the route information as well, so we know which routes creates which pools. By passing in `null` we loose this opportunity.
That is why all the EIP Processors should be refactored to have ExeuctorService as parameter. **DONE**

Let Camel keep track of created pools

Using the `DefaultExecutorServiceStrategy` we can let Camel keep track of the created pools, and thus also it can shutdown those when CamelContext is shutting down. Then Camel is handling the lifecycle for the pools it creates. And if you pass in a thread pool from an external system then you manage that lifecycle. Camel will in those cases **not** shut it down. **DONE**

Sensible defaults

The `CachedExecutorService` by the JDK is maybe a bit aggressive as its unbounded thread pool which essentially can create 1000s of threads if the sever is not busy. But end users may want to have a reasonable max size, lets say 100. So we should offer some sort of rule which you can configure what the default settings should be for thread pools created by Camel. **DONE**

Rejection policy

We should add configuration about rejection policies for new tasks submitted to a pool. The JDK has options for ABORT, RUN, WAIT, DISCARD etc. **DONE**