Asynchronous Client HTTP Transport

Asynchronous Client HTTP Transport

By default, CXF uses a transport based on the in-JDK HttpURLConnection object to perform HTTP requests. The HttpURLConnection object uses a blocking model for all IO operations which requires a per-thread execution model. From a pure performance standpoint, this model generally performs very well, but it does have problems scaling when many requests need to be executed simultaneously.

Also, the JAX-WS specification allows for generation of asynchronous methods on generated proxies as well as using asynchronous methods on the Dispatch objects. These methods can take an AsyncHandler object and return a polling Future object so applications do not have to wait for the response. With the HttpURLConnection based transport, CXF was forced to consume a background thread for each outstanding request.

Apache HttpComponents 4.x

CXF also has an HTTP client transport that is based on the Apache HTTP Components HttpAsyncClient library. Its Maven artifactId is **cxf-rt-transports-http-hc**. The HttpAsyncClient library uses a non-blocking IO model. This allows many more requests to be outstanding without consuming extra background threads. It also allows greater control over things like Keep-Alive handling which is very difficult or impossible with the HttpURLConnection based transport. However, the non-blocking model does not perform quite as well as the blocking model for pure synchronous request/response transactions.

By default, if the **cxf-rt-transports-http-hc** module is found on the classpath, CXF will use the **HttpAsyncClient** based implementation for any Async calls, but will continue to use the HttpURLConnection based transport for synchronous calls. This allows a good balance of performance for the common synchronous cases with scalability for the asynchronous cases. However, using a contextual property of "use.async.http.conduit" and set to true/false, you can control whether the async or blocking version is used. If "true", the HttpAsyncClient will be used even for synchronous calls, if "false", asynchronous calls will rely on the traditional method of using HTTPURLConnection along with a work queue to mimic the asynchronicity. And if TLSClientParameters sets an SSLSocketFactory, as SocketFactory class and SocketFactory#createSocket methods in particular are inherently blocking and sockets instantiated in such a way cannot be used for asynchronous, so this lead to use the HttpURLConnection based transport.

Another reason to use the asynchronous transport is to use HTTP methods that HttpURLConnection does not support. For example, the github.com REST API specifies the use of PATCH for some cases, but HttpURLConnection rejects PATCH.

Apache HttpComponents 5.x

Since **3.4.6**, CXF offers an HTTP client transport that is based on Apache HttpComponents HttpClient 5 library, that supports synchronous, asynchronous and reactive programming models. Its Maven artifactId is **cxf-rt-transports-http-hc5** and it serves as in-place replacement for **cxf-rt-transports-http-hc** (but the usage of those two transports together should be avoided).

> At the moment, **cxf-rt-transports-http-hc5** transport does not support OSGi based deployments

This client transport supports HTTP/2 (when enabled using **org.apache.cxf.transports.http2.enabled** property, see **Configuration** section below).

Using the HTTP Components 4.x/5.x Transport from Java Code

To force global use of the HTTP Components transport, you can set a bus-level property:

```
Bus bus = BusFactory.getDefaultBus();
// insist on the async connector to use PATCH
bus.setProperty(AsyncHTTPConduit.USE_ASYNC, Boolean.TRUE);
// allows the async connector to use HTTP/2 protocol (if supported by the server)
bus.setProperty(AsyncHTTPConduit.ENABLE_HTTP2, enableHttp2);
```

Setting Credentials

The "normal" CXF/JAX-WS method of setting user credentials via the BindingProvider.USERNAME_PROPERTY/PASSWORD_PROPERTY will work with the Async transport as well. However, the HttpAsyncClient library does have some additional capabilities around NTLM that can be leveraged. In order to use that, you need to:

- Turn on the AutoRedirect and turn off the Chunking for the Conduit. This will allow CXF to cache the response in a manner that will allow the transport to keep resending the request during the authentication negotiation.
- Force the use of the Async transport even for synchronous calls

```
bp.getRequestContext().put("use.async.http.conduit", Boolean.TRUE);
bp.getRequestContext().put("org.apache.cxf.transports.http2.enabled", Boolean.TRUE);  // optionally,
enable HTTP/2
```

or using **AsyncHTTPConduit.USE_ASYNC** constant

```
bp.getRequestContext().put(AsyncHTTPConduit.USE_ASYNC, Boolean.TRUE);
bp.getRequestContext().put(AsyncHTTPConduit.ENABLE_HTTP2, Boolean.TRUE);  // optionally, enable HTTP/2
```

- Set the property "org.apache.http.auth.Credentials" to an instance of the Credentials. For example:

```
Credentials creds = new NTCredentials("username", "pswd", null, "domain");
bp.getRequestContext().put(Credentials.class.getName(), creds);
```

Instrumenting Response Processing

In certain circumstances, it is beneficial to wrap (or instrument) the async client transport response processing. Starting from Apache CXF **4.0.4 / 3.6.3 / 3.5.8** releases, it is now possible using **AsyncHttpResponseWrapperFactory** bus extension, for example:

```
 final AsyncHttpResponseWrapper wrapper = new AsyncHttpResponseWrapper() {
     @Override
     public void responseReceived(HttpResponse response, Consumer<HttpResponse> delegate) {
         delegate.accept(response);
     }
 };

bus.setExtension(() -> wrapper, AsyncHttpResponseWrapperFactory.class);
...
```

It is very important for the **AsyncHttpResponseWrapper** to pass the call to `delegate` in order to resume the response processing chain, otherwise the response processing may never finish.

Netty 4.x

Apache CXF also offers an HTTP client transport that is based on Netty 4.x. Its Maven artifactId is **cxf-rt-transports-http-netty-client**.

This client transport supports HTTP/2 (when enabled using **org.apache.cxf.transports.http2.enabled** property, see **Configuration** section below).

Configuration

The Asynchronous HTTP Transport has several options that can set using Bus properties or via the OSGi configuration services to control various aspects of the underlying Apache HTTP Components HttpAsyncClient objects.

Settings related to the underlying TCP socket (see java.net.Socket for a definition of these values):

org.apache.cxf.transport.http.async.TCP_NODELAY (Default true)
org.apache.cxf.transport.http.async.SO_KEEPALIVE
org.apache.cxf.transport.http.async.SO_LINGER
org.apache.cxf.transport.http.async.SO_TIMEOUT

Settings related to Keep-Alive connection management:

org.apache.cxf.transport.http.async.CONNECTION_TTL	Maximum time a connection to live(from creation to expiry) . Default is 60000.
org.apache.cxf.transport.http.async.MAX_CONNECTIONS	Maximum number of connections opened in total. Default is 5000.
org.apache.cxf.transport.http.async.MAX_PER_HOST_CONNECTIONS	Maximum number of connections opened per host. Default is 1000.

Settings related to **HTTP/2** support (Apache CXF versions **4.0.2+**/**3.6.1+**/**3.5.7**+/**3.4.11+**):

org.apache.cxf.transports.http2.enabled	true | false	Allows HTTP/2 protocol if supported by the server. Default is **false**.

Settings related to Apache HttpAsyncClient threads and selectors:

org.apache.cxf.transport.http.async.ioThreadCount	Number of threads HttpAsyncClient uses to process IO events. Default is "-1" which means one thread per CPU core.
org.apache.cxf.transport.http.async.interestOpQueued **(*)**	true/false for whether the interest ops are queues or process directly.
org.apache.cxf.transport.http.async.selectInterval	Default 1000 ms. How often the selector thread wakes up if there are no events to process additional things like queue expirations.

***** - for Apache HttpComponents 4.x only

Setting to control which conduit is used

org.apache.cxf.transport.http.async.usePolicy	ALWAYS, ASYNC_ONLY, NEVER.	Similar in meaning to the "use.async.http.conduit" context property described above. Whether to use the HttpAsyncClient: ALWAYS for both synchronous and asynchronous calls, ASYNC_ONLY (default) for asynchronous calls only, NEVER will use HTTPURLConnection for both types of calls.