AEM Invoke API – REST service using HTTP Client factory

Kiran Sg AEM 8th Nov 202123rd Apr 2023 8 Minutes

Problem Statement:

The article addresses the problem of making REST-based calls from AEM to an external system and the best way to handle HTTP requests.

Requirement:

The article discusses the implementation of an OSGi-based REST service in AEM that integrates with an external system using the HTTP Client factory. The author provides detailed steps on how to create a new Apache Closable HTTP Client, prepare request configuration, pool HTTP connections, and use default headers and keepAlive strategy to execute requests.

Create an OSGi based REST service to integrate AEM with the external system, and also provide config to provide endpoint options and client factory configurations.

Introduction:

As we all know AEM is REST based Web application, however, is there a way to integrate OSGi based service to make calls to the external system.

After going through the ACS commons based HTTP Client factory, I created a more feature-friendly and rich HTTP client factory.

Create HTTPClientFactory Service Interface:

This service provides the implementation for most of the basic HTTP REST based operations like GET, PUT, POST, and DELETE operations.

```java
package com.example.core.services;

import org.apache.http.client.fluent.Executor;
import org.apache.http.client.fluent.Request;

/**
 * Factory for building pre-configured HttpClient Fluent Executor and Request objects
 * based a configure host, port and (optionally) username/password.
 * Factories will generally be accessed by service lookup using the factory.name property.
 */
public interface HttpClientFactory {

    /**
     * Get the configured Executor object from this factory.
     *
     * @return an Executor object
     */
    Executor getExecutor();

    /**
     * Create a GET request using the base hostname and port defined in the factory configuration.
     *
     * @param partialUrl the portion of the URL after the port (and slash)
     *
     * @return a fluent Request object
     */
    Request get(String partialUrl);

    /**
     * Create a PUT request using the base hostname and port defined in the factory configuration.
     *
     * @param partialUrl the portion of the URL after the port (and slash)
     *
     * @return a fluent Request object
     */
    Request put(String partialUrl);

    /**
     * Create a POST request using the base hostname and port defined in the factory configuration.
     *
```

```
41          * @param partialUrl the portion of the URL after the port (and slash)
42          *
43          * @return a fluent Request object
44          */
45         Request post(String partialUrl);
46
47         /**
48          * Create a DELETE request using the base hostname and port defined in the factory configuration.
49          *
50          * @param partialUrl the portion of the URL after the port (and slash)
51          *
52          * @return a fluent Request object
53          */
54         Request delete(String partialUrl);
55
56         /**
57          * Create a OPTIONS request using the base hostname and port defined in the factory configuration.
58          *
59          * @param partialUrl the portion of the URL after the port (and slash)
60          *
61          * @return a fluent Request object
62          */
63         Request options(String partialUrl);
64
65         /**
66          * Get External URI type is form the factory configuration.
67          *
68          * @return External URI Type
69          */
70         String getExternalURIType();
71
72         /**
73          * Get apiStoreLocatorHostName URI type is form the factory configuration.
74          *
75          * @return API StoreLocatorHost
76          */
77         String getApiStoreLocatorHostName();
78
79         Request postWithAbsolute(String absolutelUrl);
80     }
```

Create HTTPClientFactoryConfig:

Add the required attributes to create the HTTPCLientFactory.

```
1  package com.example.services.config;
2
3  import org.osgi.service.metatype.annotations.AttributeDefinition;
4  import org.osgi.service.metatype.annotations.AttributeType;
5  import org.osgi.service.metatype.annotations.ObjectClassDefinition;
6  import com.example.constants.Constants;
7
8  @ObjectClassDefinition(name = "Http Client API Configuration", description = "Http Client API Configuration")
9  public @interface HttpClientFactoryConfig {
10
11     @AttributeDefinition(name = "API Host Name", description = "API host name, e.g. https://example.com (https:/
12     String apiHostName() default Constants.DEFAULT_API_HOST_NAME;
13
14     @AttributeDefinition(name = "API URI Type Path", description = "API URI type path, e.g. /services/int/v2", ty
15     String uriType() default Constants.DEFAULT_API_URI_TYPE_PATH;
16
17     @AttributeDefinition(name = "API URI Type Path", description = "API URI type path, e.g. /services/ext/v2", ty
18     String uriExternalType() default Constants.DEFAULT_API_URI_EXTERNAL_TYPE_PATH;
19
20     @AttributeDefinition(name = "Relaxed SSL", description = "Defines if self-certified certificates should be a
21     boolean relaxedSSL() default Constants.DEFAULT_RELAXED_SSL;
22
23     @AttributeDefinition(name = "Store Locator API Host Name", description = "Store Locator API host name, e.g. h
24     String apiStoreLocatorHostName() default Constants.DEFAULT_STORE_LOCATOR_API_HOST_NAME;
25
26     @AttributeDefinition(name = "Maximum number of total open connections", description = "Set maximum number of
27     int maxTotalOpenConnections() default Constants.DEFAULT_MAXIMUM_TOTAL_OPEN_CONNECTION;
28
29     @AttributeDefinition(name = "Maximum number of concurrent connections per route", description = "Set the maxi
30     int maxConcurrentConnectionPerRoute() default Constants.DEFAULT_MAXIMUM_CONCURRENT_CONNECTION_PER_ROUTE;
31
32     @AttributeDefinition(name = "Default Keep alive connection in seconds", description = "Default Keep alive con
33     int defaultKeepAliveconnection() default Constants.DEFAULT_KEEP_ALIVE_CONNECTION;
34
35     @AttributeDefinition(name = "Default connection timeout in seconds", description = "Default connection timout
36     long defaultConnectionTimeout() default Constants.DEFAULT_CONNECTION_TIMEOUT;
37
38     @AttributeDefinition(name = "Default socket timeout in seconds", description = "Default socket timeout in sec
39     long defaultSocketTimeout() default Constants.DEFAULT_SOCKET_TIMEOUT;
40
```

```
41      @AttributeDefinition(name = "Default connection request timeout in seconds", description = "Default connecti
42      long defaultConnectionRequestTimeout() default Constants.DEFAULT_CONNECTION_REQUEST_TIMEOUT;
```

Create HttpClientFactoryImpl Service implementation:

This provides the implementation class for HTTPClientFactory Service and during @Activate/@Modified we are trying to create a new Apache Closable HTTP Client using OSGi based HttpClientBuilderFactory.

HTTP client is like a dish, and you can taste it better if your recipe is great and if you prepare it well, before making calls to the external system.

Close all Connections:

Make sure to close the existing connection if any after bundle gets activated or modified

Preparing Request Configuration:

Create Request Config Object and set **Connection timeout**, **socket timeout,** and **request timeout** based on the service configurations

Pooling HTTP Connection:

PoolingHttpClientConnectionManager maintains a pool of **HttpClientConnections** and is able to service connection requests from multiple execution threads. Connections are pooled on a per route basis. A request for a route that already the manager has persistent connections for available in the pool will be serviced by leasing a connection from the pool rather than creating a brand new connection.

Hence set the **max pool size** and number default **max per route** (per endpoint)

Things to be aware of before pooling connection is, are you making HTTPS calls to the external system if yes? Then create an **SSLConnectionSocketFactory** with **NOOP** based verifier and add all the **trusted certificates**.

Default Keep Alive Strategy:

If the Keep-Alive header is not present in the response, HttpClient assumes the connection can be kept alive indefinitely. However, many HTTP servers in general use are configured to drop persistent connections after a certain period of inactivity to conserve system resources, often without informing the client. In case the default strategy turns out to be too optimistic, one may want to provide a custom keep-alive strategy.

HTTP Client Builder OSGi Service:

Get the reference to OSGi-based httpClientBuilderFactory service, prepare a new builder, set the request configuration, and add a connection manager with a pooling connection.

Add default headers and keepAlive strategy, so that we don't have to create a new connection

Finally, create the HTTP Client out of this builder and set the client to Apache fluent Executor.

the fluent executor makes an arbitrary HttpClient instance and executes the request.

```
1  package com.example.core.services.impl;
2
3  import java.io.IOException;
4  import java.security.KeyManagementException;
5  import java.security.KeyStoreException;
6  import java.security.NoSuchAlgorithmException;
7  import java.util.ArrayList;
8  import java.util.List;
9  import java.util.concurrent.TimeUnit;
10 import org.apache.commons.lang3.StringUtils;
11 import org.apache.http.Header;
12 import org.apache.http.HttpResponse;
13 import org.apache.http.client.config.RequestConfig;
14 import org.apache.http.client.fluent.Executor;
15 import org.apache.http.client.fluent.Request;
16 import org.apache.http.config.Registry;
17 import org.apache.http.config.RegistryBuilder;
18 import org.apache.http.conn.ConnectionKeepAliveStrategy;
19 import org.apache.http.conn.socket.ConnectionSocketFactory;
20 import org.apache.http.conn.socket.PlainConnectionSocketFactory;
21 import org.apache.http.conn.ssl.NoopHostnameVerifier;
22 import org.apache.http.conn.ssl.SSLConnectionSocketFactory;
23 import org.apache.http.conn.ssl.TrustAllStrategy;
24 import org.apache.http.impl.client.CloseableHttpClient;
25 import org.apache.http.impl.client.HttpClientBuilder;
26 import org.apache.http.impl.conn.PoolingHttpClientConnectionManager;
27 import org.apache.http.message.BasicHeader;
28 import org.apache.http.osgi.services.HttpClientBuilderFactory;
29 import org.apache.http.protocol.HttpContext;
30 import org.apache.http.ssl.SSLContextBuilder;
31 import org.osgi.service.component.annotations.Activate;
32 import org.osgi.service.component.annotations.Component;
33 import org.osgi.service.component.annotations.Deactivate;
34 import org.osgi.service.component.annotations.Modified;
35 import org.osgi.service.component.annotations.Reference;
36 import org.osgi.service.metatype.annotations.Designate;
37 import org.slf4j.Logger;
38 import org.slf4j.LoggerFactory;
39 import com.example.core.services.HttpClientFactory;
40 import com.example.core.services.config.HttpClientFactoryConfig;
```

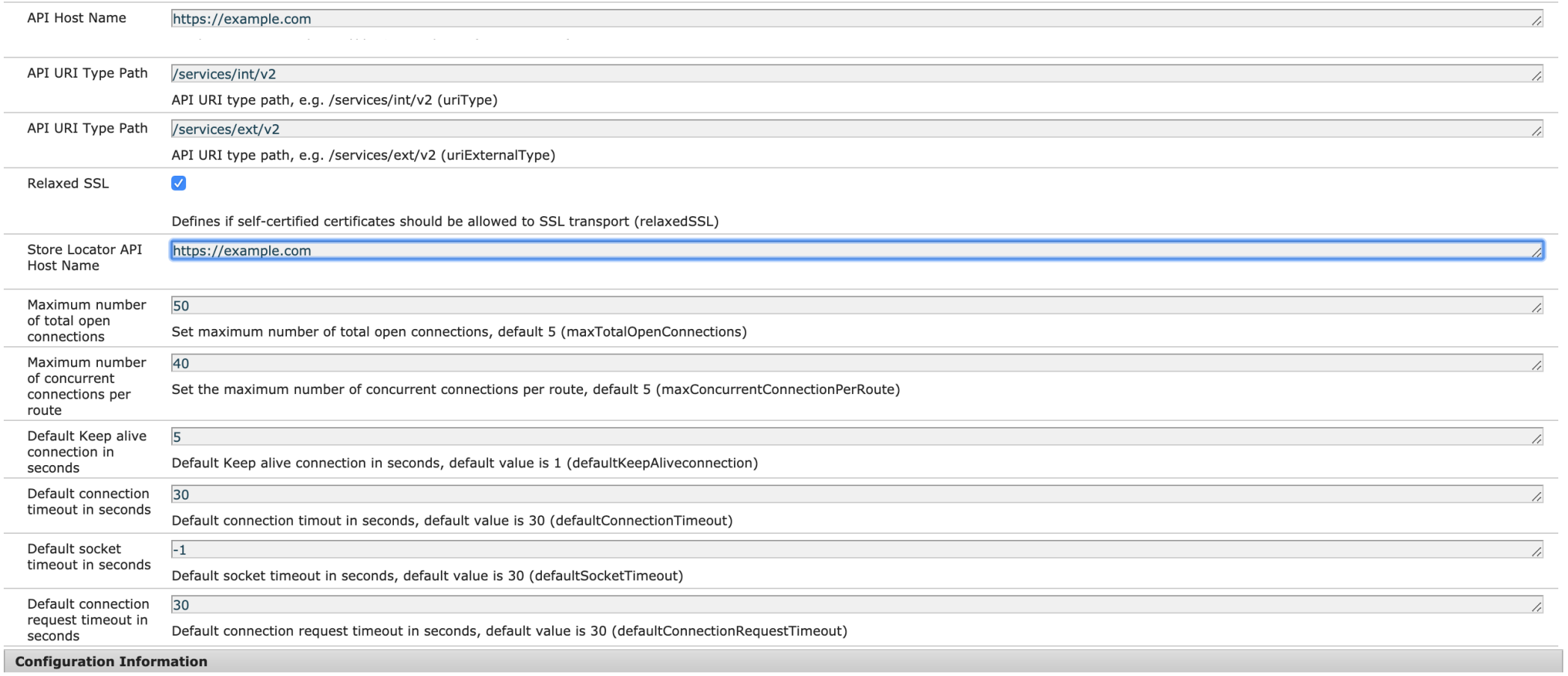
```
41
42  /**
43   * Implementation of @{@link HttpClientFactory}.
44   * <p>
45   * HttpClientFactory provides service to handle API connection and executor.
46   */
47  @Component(service = HttpClientFactory.class)
48  @Designate(ocd = HttpClientFactoryConfig.class)
49  public class HttpClientFactoryImpl implements HttpClientFactory {
50
51      private static final Logger log = LoggerFactory.getLogger(HttpClientFactoryImpl.class);
52
53      private Executor executor;
54      private String baseUrl;
55      private String uriExternalType;
56      private String apiStoreLocatorHostName;
57      private CloseableHttpClient httpClient;
58      private HttpClientFactoryConfig config;
59
60      @Reference
61      private HttpClientBuilderFactory httpClientBuilderFactory;
62
63      @Activate
64      @Modified
65      protected void activate(HttpClientFactoryConfig config) throws Exception {
66
67          log.info("############ OSGi Configs Start ################");
68          log.info("API Host Name : {}", config.apiHostName());
69          log.info("URI Type: {}", config.uriType());
70          log.info("############ OSGi Configs End ################");
71
72          closeHttpConnection();
73
74          this.config = config;
75          if (this.config.apiHostName() == null) {
76              log.debug("Configuration is not valid. Both hostname is mandatory.");
77              throw new IllegalArgumentException("Configuration is not valid. Both hostname is mandatory.");
78          }
79
80          this.uriExternalType = this.config.uriExternalType();
```

```
81          this.apiStoreLocatorHostName = this.config.apiStoreLocatorHostName();
82          this.baseUrl = StringUtils.join(this.config.apiHostName(), config.uriType());
83
84          initExecutor();
85      }
86
87      private void initExecutor() throws Exception {
88
89          PoolingHttpClientConnectionManager connMgr = null;
90
91          RequestConfig requestConfig = initRequestConfig();
92
93          HttpClientBuilder builder = httpClientBuilderFactory.newBuilder();
94          builder.setDefaultRequestConfig(requestConfig);
95
96          if (config.relaxedSSL()) {
97
98              connMgr = initPoolingConnectionManagerWithRelaxedSSL();
99
100         } else {
101
102             connMgr = new PoolingHttpClientConnectionManager();
103         }
104
105         connMgr.closeExpiredConnections();
106
107         connMgr.setMaxTotal(config.maxTotalOpenConnections());
108         connMgr.setDefaultMaxPerRoute(config.maxConcurrentConnectionPerRoute());
109
110         builder.setConnectionManager(connMgr);
111
112         List<Header> headers = new ArrayList<>();
113         headers.add(new BasicHeader("Accept", "application/json"));
114         builder.setDefaultHeaders(headers);
115         builder.setKeepAliveStrategy(keepAliveStratey);
116
117         httpClient = builder.build();
118
119         executor = Executor.newInstance(httpClient);
120     }
```

```
121
122     private PoolingHttpClientConnectionManager initPoolingConnectionManagerWithRelaxedSSL()
123             throws NoSuchAlgorithmException, KeyStoreException, KeyManagementException {
124
125         PoolingHttpClientConnectionManager connMgr;
126         SSLContextBuilder sslbuilder = new SSLContextBuilder();
127         sslbuilder.loadTrustMaterial(new TrustAllStrategy());
128         SSLConnectionSocketFactory sslsf = new SSLConnectionSocketFactory(sslbuilder.build(),
129                 NoopHostnameVerifier.INSTANCE);
130         Registry<ConnectionSocketFactory> socketFactoryRegistry = RegistryBuilder.<ConnectionSocketFactory>creat
131                 .register("http", PlainConnectionSocketFactory.getSocketFactory()).register("https", sslsf).bui
132         connMgr = new PoolingHttpClientConnectionManager(socketFactoryRegistry);
133         return connMgr;
134     }
135
136     private RequestConfig initRequestConfig() {
137
138         return RequestConfig.custom()
139                 .setConnectTimeout(Math.toIntExact(TimeUnit.SECONDS.toMillis(config.defaultConnectionTimeout())
140                 .setSocketTimeout(Math.toIntExact(TimeUnit.SECONDS.toMillis(config.defaultSocketTimeout())))
141                 .setConnectionRequestTimeout(
142                         Math.toIntExact(TimeUnit.SECONDS.toMillis(config.defaultConnectionRequestTimeout())))
143                 .build();
144     }
145
146     @Deactivate
147     protected void deactivate() {
148         closeHttpConnection();
149     }
150
151     private void closeHttpConnection() {
152         if (null != httpClient) {
153             try {
154                 httpClient.close();
155             } catch (final IOException exception) {
156                 log.debug("IOException while clossing API, {}", exception.getMessage());
157             }
158         }
159     }
160
```

```
161         @Override
162         public Executor getExecutor() {
163             return executor;
164         }
165 
166         @Override
167         public Request get(String partialUrl) {
168             String url = baseUrl + partialUrl;
169             return Request.Get(url);
170         }
171 
172         @Override
173         public Request post(String partialUrl) {
174             String url = baseUrl + partialUrl;
175             return Request.Post(url);
176         }
177 
178         @Override
179         public Request postWithAbsolute(String absolutelUrl) {
180             return Request.Post(absolutelUrl);
181         }
182 
183         @Override
184         public Request put(String partialUrl) {
185             String url = baseUrl + partialUrl;
186             return Request.Put(url);
187         }
188 
189         @Override
190         public Request delete(String partialUrl) {
191             String url = baseUrl + partialUrl;
192             return Request.Delete(url);
193         }
194 
195         @Override
196         public Request options(String partialUrl) {
197             String url = baseUrl + partialUrl;
198             return Request.Options(url);
199         }
200 
```

```
201         @Override
202         public String getExternalURIType() {
203             return uriExternalType;
204         }
205 
206         @Override
207         public String getApiStoreLocatorHostName() {
208             return apiStoreLocatorHostName;
209         }
210 
211         ConnectionKeepAliveStrategy keepAliveStratey = new ConnectionKeepAliveStrategy() {
212 
213             @Override
214             public long getKeepAliveDuration(HttpResponse response, HttpContext context) {
215                 /*
216                  * HeaderElementIterator headerElementIterator = new BasicHeaderElementIterator(
217                  * response.headerIterator(HTTP.CONN_KEEP_ALIVE));
218                  *
219                  * while (headerElementIterator.hasNext()) { HeaderElement headerElement =
220                  * headerElementIterator.nextElement(); String param = headerElement.getName();
221                  * String value = headerElement.getValue(); if (value != null &&
222                  * param.equalsIgnoreCase("timeout")) { return
223                  * TimeUnit.SECONDS.toMillis(Long.parseLong(value)); } }
224                  */
225 
226                 return TimeUnit.SECONDS.toMillis(config.defaultKeepAliveconnection());
227             }
228         };
```



OSGi configuration

References:

https://github.com/kiransg89/AEM-REST-Integration (https://github.com/kiransg89/AEM-REST-Integration)

How to use HTTP Client Factory?

— Check this out URI (https://kirantech58867409.wordpress.com/2021/11/08/aem-invoke-api-how-to-use-http-client-factory/)

Tagged:
AEM,
AEM as a Cloud Service,
aemaacs,
Best practices,
external system,
fluent executor,
HTTP Client factory,
HttpClientBuilderFactory,
HttpClientFactory,
HTTPClientFactory Service,
Keep-Alive header,
OSGi,
PoolingHttpClientConnectionManager,
Request Config Object,
REST

Published by Kiran Sg

My name is Kiran SG. I am currently working as an AEM architect and handling site revamp and migration. I try to learn new tricks and techniques every day to improve my coding and deliver value-added projects. All the tricks and techniques sometimes seem to be straightforward working code with best practices. But sometimes it won't satisfy the developer's needs. But most of the time code is either lagging to meet the requirement or doesn't follow best practices. Hence I am taking the initiative to put all my learnings and tricks in my blog series to share the working code with best practices. Hoping to help other developers like me. I have been working on AEM for 8 years now. I started my career as a Java developer. I have delivered many projects with a core customer-centric and strong focus on infrastructure and architecture. If you have any queries related to my blog topics please do connect with me directly on my email and also you can connect with me over LinkedIn and Facebook. View all posts by Kiran Sg

2 thoughts on "AEM Invoke API – REST service using HTTP Client factory"

Pingback: AEM Invoke API – How to use HTTP Client Factory – AEM best practices and development guide
Pingback: Exporting AEM Experience Fragment/Page Content for A/B Testing and External Systems without HTML Tags – AEM best practices and development guide