Reference urls

|  |  |
| --- | --- |
| main root url | <https://hc.apache.org/httpcomponents-client-4.5.x/index.html> |
| full tutorial for reading | <https://hc.apache.org/httpcomponents-client-4.5.x/current/tutorial/html/index.html> |
|  |  |

Sfd

cd

HTTP connection manger

Same like jdbc connection object and pool – to reuse a connection object after creating a connection and using instead deleting or garbage collecting that obj , we will just return that object to the pool

Same like that here also we will maintain the http connection pool, once we use , we will return back that object to the pool instead of gc ,

### All the pool is maintained by  Pooling connection manager

HTTP connections can only be used by one execution thread at a time

1 connection = is blocked by 1 thread

PoolingHttpClientConnectionManager cm = new PoolingHttpClientConnectionManager();

// Increase max total connection to 200

cm.setMaxTotal(200);

// Increase default max connection per route to 20

cm.setDefaultMaxPerRoute(20);

// Increase max connections for localhost:80 to 50

HttpHost localhost = new HttpHost("locahost", 80);

cm.setMaxPerRoute(new HttpRoute(localhost), 50);

CloseableHttpClient httpClient = HttpClients.custom()

.setConnectionManager(cm)

.build();

### 2.3.4. Connection manager shutdown

When an HttpClient instance is no longer needed and is about to go out of scope it is important to shut down its connection manager to ensure that all connections kept alive by the manager get closed and system resources allocated by those connections are released.

CloseableHttpClient httpClient = <...>

httpClient.close();

## 2.5. Connection eviction policy

This means

|  |  |
| --- | --- |
| current way | additional cfg |
| when we execute a http req via pool,  internally before making a request apache will check if conn taken from pool is stale or not (stale means a conn no longer valid because it was closed on the server side) |  |

The stale connection check is not 100% reliable. The only feasible solution that does not involve a one thread per socket model for idle connections is a dedicated monitor thread used to evict connections that are considered expired due to a long period of inactivity. The monitor thread can periodically call ClientConnectionManager#closeExpiredConnections() method to close all expired connections and evict closed connections from the pool. It can also optionally call ClientConnectionManager#closeIdleConnections() method to close all connections that have been idle over a given period of time.

public static class IdleConnectionMonitorThread extends Thread {

private final HttpClientConnectionManager connMgr;

private volatile boolean shutdown;

public IdleConnectionMonitorThread(HttpClientConnectionManager connMgr) {

super();

this.connMgr = connMgr;

}

@Override

public void run() {

try {

while (!shutdown) {

synchronized (this) {

wait(5000);

// Close expired connections

connMgr.closeExpiredConnections();

// Optionally, close connections

// that have been idle longer than 30 sec

connMgr.closeIdleConnections(30, TimeUnit.SECONDS);

}

}

} catch (InterruptedException ex) {

// terminate

}

}

public void shutdown() {

shutdown = true;

synchronized (this) {

notifyAll();

}

}

}

When client will face stale connection exception

* Many real-world HTTP servers will **silently drop idle connections** after some timeout (say 5–60 seconds) to free resources.
* But they don’t always tell the client!

So the client might try to reuse a connection that the server has already closed → leading to:

* Connection reset or SocketException: Connection reset by peer errors.

Attach keystore, cacert files and build client

Here now we are loading both keystore and truststore and attaching to ssl context

|  |  |
| --- | --- |
| import org.apache.http.conn.ssl.SSLConnectionSocketFactory;  import org.apache.http.impl.conn.PoolingHttpClientConnectionManager;  import org.apache.http.impl.client.CloseableHttpClient;  import org.apache.http.impl.client.HttpClients;  import org.apache.http.conn.ssl.TrustSelfSignedStrategy;  import org.apache.http.ssl.SSLContextBuilder;  import javax.net.ssl.SSLContext;  import java.io.File;  import java.security.KeyStore;  public class SslHttpClientExample {  public static void main(String[] args) throws Exception {  String keystorePath = "client-keystore.jks";  String keystorePassword = "changeit";  String truststorePath = "client-truststore.jks";  String truststorePassword = "changeit";  // Load KeyStore  KeyStore keyStore = KeyStore.getInstance("JKS");  keyStore.load(new java.io.FileInputStream(keystorePath), keystorePassword.toCharArray());  // Load TrustStore  KeyStore trustStore = KeyStore.getInstance("JKS");  trustStore.load(new java.io.FileInputStream(truststorePath), truststorePassword.toCharArray());  // Create SSLContext with both KeyStore and TrustStore  SSLContext sslContext = SSLContextBuilder.create()  .loadKeyMaterial(keyStore, keystorePassword.toCharArray())  .loadTrustMaterial(trustStore, new TrustSelfSignedStrategy())  .build();  // Create SSLConnectionSocketFactory with the SSLContext  SSLConnectionSocketFactory sslSocketFactory = new SSLConnectionSocketFactory(sslContext);  // Create connection manager using the SSLConnectionSocketFactory  PoolingHttpClientConnectionManager connManager =  new PoolingHttpClientConnectionManager(  org.apache.http.config.RegistryBuilder.<org.apache.http.conn.socket.ConnectionSocketFactory>create()  .register("https", sslSocketFactory)  .register("http", org.apache.http.conn.socket.PlainConnectionSocketFactory.getSocketFactory())  .build());  or in realtime we will use pooling HttpClientCon manager only  ConnectionSocketFactory plainsf = <...>  LayeredConnectionSocketFactory sslsf = <...>  Registry<ConnectionSocketFactory> r = RegistryBuilder.<ConnectionSocketFactory>create()  .register("http", plainsf)  .register("https", sslsf)  .build();  HttpClientConnectionManager cm = new PoolingHttpClientConnectionManager(r);  HttpClients.custom()  .setConnectionManager(cm)  .build();  // Optional: tune pool size  connManager.setMaxTotal(50);  connManager.setDefaultMaxPerRoute(10);  // Build HttpClient with custom connection manager  CloseableHttpClient httpClient = HttpClients.custom()  .setSSLSocketFactory(sslSocketFactory)  .setConnectionManager(connManager)  .build();  // Now you can use `httpClient` to execute HTTPS requests  System.out.println("HttpClient with custom SSLContext created successfully."); |  |