记录 elasticsearch6.2.2-ssl- 身份认证 -springboot

作者：alanfans
原文链接：https://ld246.com/article/1521542013452
来源网站：链滴

elasticsearch

参考资料:

https://github.com/nicoraynaud/rubiks-spring-rest-elasticsearch/

https://www.journaldev.com/18148/spring-boot-elasticsearch#spring-boot-elasticsearch-6

http://blog.csdn.net/qq_34246546/article/details/78919744

http://blog.csdn.net/hololens/article/details/79594583

很不幸的是，spring-data-elasticsearch支持elasticsearch版本太低

有两个方案: elasticsearch-rest-high-level-client和elasticsearch-rest-client-sniffer,本文用的elasti search-rest-high-level-client

遇到错误有

General SSLEngine problem

解: https://stackoverflow.com/questions/20988183/certificateexception-with-async-http-clien -for-https

最后新增成功

代码

github: https://github.com/twalanfans/renren-fast02.git

pom.xml

```
        <dependency>
          <groupId>org.elasticsearch</groupId>
          <artifactId>elasticsearch</artifactId>
          <version>6.2.2</version>
        </dependency>
        <dependency>
          <groupId>org.elasticsearch.client</groupId>
          <artifactId>transport</artifactId>
          <version>6.2.2</version>
        </dependency>
        <dependency>
          <groupId>org.elasticsearch.client</groupId>
          <artifactId>elasticsearch-rest-high-level-client</artifactId>
          <version>6.2.2</version>
         </dependency>
        <dependency>
```

```
import java.security.KeyManagementException;
import java.security.KeyStoreException;
import java.security.NoSuchAlgorithmException;
import java.security.cert.CertificateException;
import java.security.cert.X509Certificate;

import javax.net.ssl.SSLContext;

import org.apache.http.HttpHost;
import org.apache.http.auth.AuthScope;
import org.apache.http.auth.UsernamePasswordCredentials;
import org.apache.http.client.CredentialsProvider;
import org.apache.http.client.config.RequestConfig;
import org.apache.http.impl.client.BasicCredentialsProvider;
import org.apache.http.impl.nio.client.HttpAsyncClientBuilder;
import org.apache.http.ssl.SSLContextBuilder;
import org.apache.http.ssl.TrustStrategy;
import org.elasticsearch.client.RestClient;
import org.elasticsearch.client.RestClientBuilder;
import org.elasticsearch.client.RestHighLevelClient;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.beans.factory.config.AbstractFactoryBean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class ElasticSearchConfiguration extends AbstractFactoryBean {
    private static final Logger LOG = LoggerFactory.getLogger(ElasticSearchConfiguration.class)


    @Value("${spring.data.elasticsearch.cluster-host}")
    private String clusterHost;

    @Value("${spring.data.elasticsearch.cluster-port}")
```

```
    private int clusterPort;

    /**
     * 用户名
     */
    @Value("${spring.data.elasticsearch.username}")
    private String userName;

    /**
     * 密码
     */
    @Value("${spring.data.elasticsearch.password}")
    private String password;

    private static RestClientBuilder builder;
    private static RestClient restClient;
    private static RestHighLevelClient restHighLevelClient;

    @Override
    public void destroy() {
        try {
            if (restHighLevelClient != null) {
                restHighLevelClient.close();
            }
        } catch (final Exception e) {
            LOG.error("Error closing ElasticSearch client: ", e);
        }
    }

    @Override
    public Class<RestHighLevelClient> getObjectType() {
        return RestHighLevelClient.class;
    }

    @Override
    public boolean isSingleton() {
        return false;
    }

    @Override
    public RestHighLevelClient createInstance() {
        return buildClient();
    }


    private RestHighLevelClient buildClient() {
        try {
            builder = RestClient.builder(new HttpHost(clusterHost, clusterPort, "https"));
            if(true){
                setConnectTimeOutConfig();
            }
//            if(true){
//                setMutiConnectConfig();
//            }
```

```
            restClient = builder.build();
            restHighLevelClient = new RestHighLevelClient(builder);
        } catch (Exception e) {
            LOG.error(e.getMessage());
        }
        return restHighLevelClient;
    }

 // 主要关于异步httpclient的连接延时配置
    public void setConnectTimeOutConfig(){
        final CredentialsProvider credentialsProvider = new BasicCredentialsProvider();
        credentialsProvider.setCredentials(AuthScope.ANY,new UsernamePasswordCredentials(u
erName, password));

        builder.setRequestConfigCallback(new RestClientBuilder.RequestConfigCallback() {
            @Override
            public RequestConfig.Builder customizeRequestConfig(RequestConfig.Builder builder)

                builder.setConnectTimeout(1000);
                builder.setSocketTimeout(30000);
                builder.setConnectionRequestTimeout(500);
                builder.setAuthenticationEnabled(true);
                return builder;
            }
        });
        /*
        builder.setRequestConfigCallback(requestConfigBuilder -> {
            requestConfigBuilder.setConnectTimeout(CONNECT_TIME_OUT);
            requestConfigBuilder.setSocketTimeout(SOCKET_TIME_OUT);
            requestConfigBuilder.setConnectionRequestTimeout(CONNECTION_REQUEST_TIME_
UT);
            return requestConfigBuilder;
        });
         */
        builder.setHttpClientConfigCallback(new RestClientBuilder.HttpClientConfigCallback() {

            @Override
            public HttpAsyncClientBuilder customizeHttpClient(HttpAsyncClientBuilder httpClient
uilder) {
                SSLContext sslContext;
                try {
                    sslContext = new SSLContextBuilder().loadTrustMaterial(null, new TrustStrategy()

                        //信任所有证书
                        @Override
                        public boolean isTrusted(X509Certificate[] chain, String authType) throws Certif
cateException {
                            return true;
                        }
                    }).build();
                    httpClientBuilder.setSSLContext(sslContext);
                } catch (KeyManagementException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
```

```
            } catch (NoSuchAlgorithmException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            } catch (KeyStoreException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }

            return httpClientBuilder.setDefaultCredentialsProvider(credentialsProvider);
        }

    });
}


/**
 *    主要关于异步httpclient的连接数配置
 */
public static void setMutiConnectConfig(){
    builder.setHttpClientConfigCallback(new RestClientBuilder.HttpClientConfigCallback() {
        @Override
        public HttpAsyncClientBuilder customizeHttpClient(HttpAsyncClientBuilder httpAsyncC
ientBuilder) {
            httpAsyncClientBuilder.setMaxConnTotal(100);
            httpAsyncClientBuilder.setMaxConnPerRoute(100);
            return httpAsyncClientBuilder;
        }
    });
    /*
    builder.setHttpClientConfigCallback(httpClientBuilder -> {
        httpClientBuilder.setMaxConnTotal(MAX_CONNECT_NUM);
        httpClientBuilder.setMaxConnPerRoute(MAX_CONNECT_PER_ROUTE);
        return httpClientBuilder;
    });
    */
}

public static RestClient getClient(){
    return restClient;
}

}```


import java.util.HashMap;
import java.util.Map;
import java.util.UUID;

import org.elasticsearch.ElasticsearchException;
import org.elasticsearch.action.delete.DeleteRequest;
import org.elasticsearch.action.delete.DeleteResponse;
```

```
import org.elasticsearch.action.get.GetRequest;
import org.elasticsearch.action.get.GetResponse;
import org.elasticsearch.action.index.IndexRequest;
import org.elasticsearch.action.index.IndexResponse;
import org.elasticsearch.action.update.UpdateRequest;
import org.elasticsearch.action.update.UpdateResponse;
import org.elasticsearch.client.RestHighLevelClient;
import org.elasticsearch.common.xcontent.XContentType;
import org.springframework.stereotype.Repository;

import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;

@Repository
public class ChiliDao {


private final String INDEX = "magnet";
  private final String TYPE = "Chili";
  private RestHighLevelClient restHighLevelClient;
  private ObjectMapper objectMapper;

  public ChiliDao( ObjectMapper objectMapper, RestHighLevelClient restHighLevelClient) {
    this.objectMapper = objectMapper;
    this.restHighLevelClient = restHighLevelClient;
  }


  public Chili insertChili(Chili chili){

//         chili.setId(UUID.randomUUID().toString());
Map<String,String> dataMap = objectMapper.convertValue(chili, Map.class);
IndexRequest indexRequest = new IndexRequest(INDEX, TYPE, chili.getId()).source(dataMap);
try {
IndexResponse response = restHighLevelClient.index(indexRequest);
} catch(ElasticsearchException e) {
e.getDetailedMessage();
} catch (java.io.IOException ex){
ex.getLocalizedMessage();
}
return chili;
}


  public Map<String, Object> getChiliById(String id){
```

```
    GetRequest getRequest = new GetRequest(INDEX, TYPE, id);
    GetResponse getResponse = null;
    try {
      getResponse = restHighLevelClient.get(getRequest);
    } catch (java.io.IOException e){
      e.getLocalizedMessage();
    }
    Map<String, Object> sourceAsMap = getResponse.getSourceAsMap();
    return sourceAsMap;
  }

  public Map<String, Object> updateBookById(String id, Chili chili){
    UpdateRequest updateRequest = new UpdateRequest(INDEX, TYPE, id)
          .fetchSource(true);    // Fetch Object after its update
    Map<String, Object> error = new HashMap<>();
    error.put("Error", "Unable to update chili");
    try {
      String chiliJson = objectMapper.writeValueAsString(chili);
      updateRequest.doc(chiliJson, XContentType.JSON);
      UpdateResponse updateResponse = restHighLevelClient.update(updateRequest);
      Map<String, Object> sourceAsMap = updateResponse.getGetResult().sourceAsMap();
      return sourceAsMap;
    }catch (JsonProcessingException e){
      e.getMessage();
    } catch (java.io.IOException e){
      e.getLocalizedMessage();
    }
    return error;
  }

  public void deleteBookById(String id) {
    DeleteRequest deleteRequest = new DeleteRequest(INDEX, TYPE, id);
    try {
      DeleteResponse deleteResponse = restHighLevelClient.delete(deleteRequest);
    } catch (java.io.IOException e){
      e.getLocalizedMessage();
    }
  }

}```
```