Redis异步维表

概述

- 之前和大家聊过维表，也聊过FLINK中如何使用维表，目前FLINK自带的维表有两种，一种是JDBC维表，一种是HBASE维表。其中JDBC维表支持缓存，HBASE暂不支持
- 两种维表也都是同步维表，性能较弱
- 我之前也说过会在后面的教程实现以下异步第三方维表，加上一个叫`啤酒鸭`的问我相关的问题，就抽空写了一把
- redis异步客户端我用的是`Lettuce`，大家也可以用`Redission`，`Jedis`是同步的，大家千万注意
- 不过和我之前说的一样，缓存会遇到旧数据的问题，所以也相当于给大家留了一个小作业~~（主要没时间写，写完代码和博客就快下班了）~~，通过定时器，定时更新缓存的数据，保证尽可能拿到最新的维表数据
- 下面的代码在我的github库里面都有，包括以前教程的代码也是一样

TablesFactory

- 想通过DDL来定义我们的维表的话，就必须得实现`StreamTableSourceFactory`，`StreamTableSinkFactory`其中之一或者两者全部实现。一个对应数据源表，一个对应于数据结果表，需要哪个实现哪个吧
- 之后需要实现4个方法
  - `createStreamTableSink` 创建流类型tableSink
  - `createStreamTableSource`创建流类型tableSource
  - `requiredContext` 只有DDL语句WITH里面的参数&值和该方法传递的参数完全一致，DDL才能映射到这个工厂类
  - `supportedProperties` 支持的参数&值，用于验证
- 同时，需要在`resource`目录下建`META-INF/services/org.apache.flink.table.factories.TableFactory`路径以及文件，并在文件里面写入你的工厂类全路径。主要是为了通过SPI来发现你的工厂类
- 下面贴一下代码

```
package factory;

import org.apache.flink.api.common.typeinfo.TypeInformation;
import org.apache.flink.api.java.tuple.Tuple2;
import org.apache.flink.table.api.TableSchema;
import org.apache.flink.table.descriptors.DescriptorProperties;
import org.apache.flink.table.descriptors.JDBCValidator;
import org.apache.flink.table.descriptors.SchemaValidator;
import org.apache.flink.table.factories.StreamTableSinkFactory;
import org.apache.flink.table.factories.StreamTableSourceFactory;
import org.apache.flink.table.sinks.StreamTableSink;
import org.apache.flink.table.sources.StreamTableSource;
import org.apache.flink.table.types.utils.TypeConversions;
```

```
import org.apache.flink.table.utils.TableSchemaUtils;
import org.apache.flink.types.Row;
import source.RedisLookupTableSource;
import util.RedisValidator;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import static org.apache.flink.table.descriptors.Schema.*;
import static org.apache.flink.table.descriptors.Schema.SCHEMA_NAME;
import static util.RedisValidator.*;

public class RedisTableSourceSinkFactory implements
        StreamTableSourceFactory<Row>,
        StreamTableSinkFactory<Tuple2<Boolean, Row>> {
    //数据输出使用
    @Override
    public StreamTableSink<Tuple2<Boolean, Row>> createStreamTableSink(Map<String,
String> properties) {
        throw new IllegalArgumentException("unSupport sink now");
    }

    //数据源使用，维表也算，其实感觉维表应该独立开
    @Override
    public StreamTableSource<Row> createStreamTableSource(Map<String, String>
properties) {
        //校验参数
        DescriptorProperties descriptorProperties = getValidatedProperties(properti
es);

        TableSchema schema = TableSchemaUtils.getPhysicalSchema(descriptorPropertie
s.getTableSchema(SCHEMA));

        RedisLookupTableSource.Builder builder = RedisLookupTableSource.builder()
                .setFieldNames(schema.getFieldNames())
                .setFieldTypes(schema.getFieldTypes())
                .setIp(descriptorProperties.getString(CONNECTOR_IP))
                .setPort(Integer.parseInt(descriptorProperties.getString(CONNECTOR_
PORT)));
                //当缓存相关参数为空时，不会出现异常
        descriptorProperties.getOptionalLong(CONNECTOR_LOOKUP_CACHE_MAX_ROWS).ifPre
sent(builder::setCacheMaxSize);
        descriptorProperties.getOptionalLong(CONNECTOR_LOOKUP_CACHE_TTL).ifPresent(
builder::setCacheExpireMs);

        return builder.build();

    }

    //redis维表 需要参数值是这样的
    @Override
    public Map<String, String> requiredContext() {
        Map<String, String> context = new HashMap<>();
        context.put(CONNECTOR_TYPE, CONNECTOR_TYPE_VALUE_REDIS);
        context.put(CONNECTOR_PROPERTY_VERSION, "1"); // backwards compatibility
        return context;
    }

    //需要的参数
    @Override
    public List<String> supportedProperties() {
```

```
        List<String> properties = new ArrayList<>();

        properties.add(CONNECTOR_IP);
        properties.add(CONNECTOR_PORT);
        properties.add(CONNECTOR_VERSION);
        properties.add(CONNECTOR_LOOKUP_CACHE_MAX_ROWS);
        properties.add(CONNECTOR_LOOKUP_CACHE_TTL);

        // schema
        properties.add(SCHEMA + ".#." + SCHEMA_DATA_TYPE);
        properties.add(SCHEMA + ".#." + SCHEMA_TYPE);
        properties.add(SCHEMA + ".#." + SCHEMA_NAME);

        return properties;
    }

    private DescriptorProperties getValidatedProperties(Map<String, String>
properties) {
        final DescriptorProperties descriptorProperties = new DescriptorProperties(
true);

        descriptorProperties.putProperties(properties);

        new SchemaValidator(true, false, false).validate(descriptorProperties);

        new RedisValidator().validate(descriptorProperties);

        return descriptorProperties;
    }
}
```

TableSource

- 有了工厂，下面就得有工厂的实现
- 我们需要实现`LookupableTableSource`，`StreamTableSource`这两个类，并且实现这些个方法
  - `getLookupFunction` 返回真正去redis拿数据的**工人**类（同步模式）
  - `getAsyncLookupFunction` 返回真正去redis拿数据的**工人**类（异步模式）
  - `isAsyncEnabled`是否是异步模式
  - `getTableSchema`表结构
  - `getDataStream`获取数据流，我们这只支持维表使用，所以直接返回空吧
  - `getProducedDataType`产生的数据类型

```
package source;

import lookup.RedisLookupFunction;
import org.apache.flink.api.common.typeinfo.TypeInformation;
import org.apache.flink.api.java.typeutils.RowTypeInfo;
import org.apache.flink.streaming.api.datastream.DataStream;
import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
import org.apache.flink.table.api.TableSchema;
import org.apache.flink.table.functions.AsyncTableFunction;
import org.apache.flink.table.functions.TableFunction;
import org.apache.flink.table.sources.LookupableTableSource;
import org.apache.flink.table.sources.StreamTableSource;
import org.apache.flink.table.types.DataType;
```

```
import org.apache.flink.table.types.utils.TypeConversions;
import org.apache.flink.types.Row;

public class RedisLookupTableSource implements
        LookupableTableSource<Row>, StreamTableSource<Row> {

    private final String[] fieldNames;
    private final TypeInformation[] fieldTypes;

    private final String ip;
    private final int port;

    private final long cacheMaxSize;
    private final long cacheExpireMs;

    private RedisLookupTableSource(String[] fieldNames, TypeInformation[]
fieldTypes, String ip, int port, long cacheMaxSize, long cacheExpireMs) {
        this.fieldNames = fieldNames;
        this.fieldTypes = fieldTypes;
        this.ip = ip;
        this.port = port;
        this.cacheMaxSize = cacheMaxSize;
        this.cacheExpireMs = cacheExpireMs;
    }

    //返回同步的，我们用的是异步的，这边直接返回null
    @Override
    public TableFunction getLookupFunction(String[] lookupKeys) {
        return null;
    }

    //返回异步的
    @Override
    public AsyncTableFunction getAsyncLookupFunction(String[] lookupKeys) {
        return RedisLookupFunction.builder()
                .setFieldNames(fieldNames)
                .setFieldTypes(fieldTypes)
                .setIp(ip)
                .setPort(port)
                .setCacheMaxSize(cacheMaxSize)
                .setCacheExpireMs(cacheExpireMs)
                .build();
    }

    //表示是异步
    @Override
    public boolean isAsyncEnabled() {
        return true;
    }

    //表结构
    @Override
    public TableSchema getTableSchema() {
        return TableSchema.builder()
                .fields(fieldNames, TypeConversions.fromLegacyInfoToDataType(fieldT
ypes))
                .build();
    }

    //做数据源的时候使用，我们这用不上，直接报错了
    @Override
    public DataStream<Row> getDataStream(StreamExecutionEnvironment execEnv) {
        throw new IllegalArgumentException("unSupport source now");
```

```
    }

    //数据输出结构
    @Override
    public DataType getProducedDataType() {
        return TypeConversions.fromLegacyInfoToDataType(new RowTypeInfo(fieldTypes,
fieldNames));
    }

    public static Builder builder() {
        return new Builder();
    }


    public static class Builder {
        private String[] fieldNames;
        private TypeInformation[] fieldTypes;

        private String ip;
        private int port;

        private long cacheMaxSize;
        private long cacheExpireMs;

        public Builder setFieldNames(String[] fieldNames) {
            this.fieldNames = fieldNames;
            return this;
        }

        public Builder setFieldTypes(TypeInformation[] fieldTypes) {
            this.fieldTypes = fieldTypes;
            return this;
        }

        public Builder setIp(String ip) {
            this.ip = ip;
            return this;
        }

        public Builder setPort(int port) {
            this.port = port;
            return this;
        }

        public Builder setCacheMaxSize(long cacheMaxSize) {
            this.cacheMaxSize = cacheMaxSize;
            return this;
        }

        public Builder setCacheExpireMs(long cacheExpireMs) {
            this.cacheExpireMs = cacheExpireMs;
            return this;
        }

        public RedisLookupTableSource build() {
            return new RedisLookupTableSource(fieldNames, fieldTypes, ip, port,
cacheMaxSize, cacheExpireMs);
        }
    }
}
```

LookupFunction

- 最底层人民，真正去redis拿数的**工人**类
- 因为我们是异步的方式，所以继承的`AsyncTableFunction`，同步方式继承`TableFunction`
- 需要重写3个方法
  - `open` 获取连接、初始化的地方
  - `getResultType` 返回类型
  - `close` 关闭连接
- 同时需要一个`)`这样的方法
  - 第一个参数代表是异步I/O，通过这个对象来返回数据
  - 第二个就是我们传递进来的参数值了
- 老样子，直接贴代码

```
package lookup;

import io.lettuce.core.RedisClient;
import io.lettuce.core.RedisURI;
import io.lettuce.core.api.async.RedisAsyncCommands;
import org.apache.flink.api.common.typeinfo.TypeInformation;
import org.apache.flink.api.java.typeutils.RowTypeInfo;
import org.apache.flink.shaded.guava18.com.google.common.cache.Cache;
import org.apache.flink.shaded.guava18.com.google.common.cache.CacheBuilder;
import org.apache.flink.table.functions.AsyncTableFunction;
import org.apache.flink.table.functions.FunctionContext;
import org.apache.flink.types.Row;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.util.Collection;
import java.util.Collections;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.TimeUnit;

public class RedisLookupFunction extends AsyncTableFunction<Row> {

    private static final Logger log = LoggerFactory.getLogger(RedisLookupFunction.c
lass);


    private final String[] fieldNames;
    private final TypeInformation[] fieldTypes;

    private final String ip;
    private final int port;

    private final long cacheMaxSize;
    private final long cacheExpireMs;
    private transient Cache<String, String> cache;


    //异步客户端
    private transient RedisAsyncCommands<String, String> asyncClient;
    private transient RedisClient redisClient;


    public RedisLookupFunction(String[] fieldNames, TypeInformation[] fieldTypes,
String ip, int port, long cacheMaxSize, long cacheExpireMs) {
        this.fieldNames = fieldNames;
```

```
        this.fieldTypes = fieldTypes;
        this.ip = ip;
        this.port = port;
        this.cacheMaxSize = cacheMaxSize;
        this.cacheExpireMs = cacheExpireMs;
    }

    public static Builder builder() {
        return new Builder();
    }

    public void eval(CompletableFuture<Collection<Row>> future, String key) {

        //先去缓存取，取的到就返回，取不到就从redis拿
        if (cache != null) {

            String value = cache.getIfPresent(key);
            log.info("value in cache is null?:{}", value == null);
            if (value != null) {
                future.complete(Collections.singletonList(Row.of(key, value)));
                return;
            }
        }

        //异步从redis中获取，如果redis取出为null，赋值"",防止这条key下次再来又查redis，导致
缓存雪崩
        try {
            asyncClient.get(key).thenAccept(value -> {
                if (value == null) {
                    value = "";
                }
                if (cache != null)
                    cache.put(key, value);
                future.complete(Collections.singletonList(Row.of(key, value)));
            });
        } catch (Exception e) {
            log.error("get from redis fail", e);
            throw new RuntimeException("get from redis fail", e);
        }


    }

    @Override
    public void open(FunctionContext context) throws Exception {
        try {
            //建立redis 异步客户端，我这用的是 Lettuce 也可以使用别的，随意
            RedisURI redisUri = RedisURI.builder()
                    .withHost(ip)
                    .withPort(port)
                    .build();
            redisClient = RedisClient.create(redisUri);
            asyncClient = redisClient.connect().async();
        } catch (Exception e) {
            throw new Exception("build redis async client fail", e);
        }

        try {
            //初始化缓存大小
            this.cache = cacheMaxSize <= 0 || cacheExpireMs <= 0 ? null :
CacheBuilder.newBuilder()
                    .expireAfterWrite(cacheExpireMs, TimeUnit.MILLISECONDS)
                    .maximumSize(cacheMaxSize)
```

```
                    .build();
            log.info("cache is null ? :{}", cache == null);
        } catch (Exception e) {
            throw new Exception("build cache fail", e);

        }


    }

    //返回类型
    @Override
    public TypeInformation<Row> getResultType() {
        return new RowTypeInfo(fieldTypes, fieldNames);
    }

    //扫尾工作，关闭连接
    @Override
    public void close() throws Exception {
        cache.cleanUp();
        asyncClient.shutdown(true);
        redisClient.shutdown();
        super.close();
    }

    public static class Builder {
        private String[] fieldNames;
        private TypeInformation[] fieldTypes;

        private String ip;
        private int port;

        private long cacheMaxSize;
        private long cacheExpireMs;


        public Builder setFieldNames(String[] fieldNames) {
            this.fieldNames = fieldNames;
            return this;
        }

        public Builder setFieldTypes(TypeInformation[] fieldTypes) {
            this.fieldTypes = fieldTypes;
            return this;
        }

        public Builder setIp(String ip) {
            this.ip = ip;
            return this;
        }

        public Builder setPort(int port) {
            this.port = port;
            return this;
        }

        public Builder setCacheMaxSize(long cacheMaxSize) {
            this.cacheMaxSize = cacheMaxSize;
            return this;
        }

        public Builder setCacheExpireMs(long cacheExpireMs) {
            this.cacheExpireMs = cacheExpireMs;
```

```
            return this;
        }


        public RedisLookupFunction build() {
            return new RedisLookupFunction(fieldNames, fieldTypes, ip, port,
cacheMaxSize, cacheExpireMs);
        }
    }
}
```

主类

- 感谢啤酒鸭的提醒，我才发现博客没有上传入口类代码，不过入口类代码很简单，大家一目了然

```
package tutorial;

import org.apache.flink.api.common.typeinfo.TypeInformation;
import org.apache.flink.api.common.typeinfo.Types;
import org.apache.flink.streaming.api.datastream.DataStream;
import org.apache.flink.streaming.api.datastream.SingleOutputStreamOperator;
import org.apache.flink.streaming.api.functions.source.RichParallelSourceFunction;
import org.apache.flink.table.api.Table;
import org.apache.flink.types.Row;

import static util.FlinkConstant.*;

public class FlinkSql07 {

    //DDL语句
    public static final String REDIS_TABLE_DIM_DDL = "" +
            "CREATE TABLE redis_dim (\n" +
            "first String,\n" +
            "name String\n" +
            ") WITH (\n" +
            " 'connector.type' = 'redis', \n" +
            " 'connector.ip' = '127.0.0.1', \n" +
            " 'connector.port' = '6379', \n" +
            " 'connector.lookup.cache.max-rows' = '10', \n" +
            " 'connector.lookup.cache.ttl' = '10000000', \n" +
            " 'connector.version' = '2.6' \n" +
            ")";

    public static void main(String[] args) throws Exception {

        DataStream<Row> ds = env.addSource(new RichParallelSourceFunction<Row>() {

            volatile boolean flag = true;

            @Override
            public void run(SourceContext<Row> ctx) throws Exception {
                while (flag) {
                    Row row = new Row(2);
                    row.setField(0, 1);
                    row.setField(1, "a");
                    ctx.collect(row);
                    Thread.sleep(1000);
                }
```

```
                }

                @Override
                public void cancel() {
                    flag = false;
                }
            }).returns(Types.ROW(Types.INT, Types.STRING));

            //注册redis维表
            tEnv.sqlUpdate(REDIS_TABLE_DIM_DDL);

            //source注册成表
            tEnv.createTemporaryView("test", ds, "id,first,p.proctime");

            //join语句
            Table table = tEnv.sqlQuery("select a.*,b.* from test a left join redis_dim
    FOR SYSTEM_TIME AS OF a.p AS b on a.first = b.first");

            //输出
            tEnv.toAppendStream(table, Row.class).print("FlinkSql07");

            env.execute("FlinkSql07");


        }
    }
```

写在最后

- 代码里面都有详细的注释，配合FLINK源码食用更加
- 参考了JDBC维表的实现方式，包括缓存部分
- 缓存实现的方式有多种，我这用的是GUAVA的LRU，写死在这了。大家可以根据DDL，来生成指定的缓存方式
- 最后麻烦大家点赞关注~~~

pom.xml

```
        <?xml version="1.0" encoding="UTF-8"?>
        <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org
    /2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
    maven.apache.org/xsd/maven-4.0.0.xsd">
            <modelVersion>4.0.0</modelVersion>

            <groupId>org.example</groupId>
            <artifactId>flink-tech</artifactId>
            <version>1.0-SNAPSHOT</version>

            <properties>
                <flink.version>1.10.0</flink.version>
                <scala.binary.version>2.11</scala.binary.version>
            </properties>

            <dependencies>
                <!-- Flink modules -->
                <dependency>
                    <groupId>org.apache.flink</groupId>
```

```
                    <artifactId>flink-table-api-java</artifactId>
                    <version>${flink.version}</version>
                    <scope>provided</scope>
                </dependency>
                <dependency>
                    <groupId>org.apache.flink</groupId>
                    <artifactId>flink-table-planner-blink_${scala.binary.version}</arti
factId>
                    <version>${flink.version}</version>

                    <scope>provided</scope>
                    <exclusions>
                        <exclusion>
                            <artifactId>scala-library</artifactId>
                            <groupId>org.scala-lang</groupId>
                        </exclusion>
                        <exclusion>
                            <artifactId>slf4j-api</artifactId>
                            <groupId>org.slf4j</groupId>
                        </exclusion>
                    </exclusions>
                </dependency>
                <dependency>
                    <groupId>org.apache.flink</groupId>
                    <artifactId>flink-json</artifactId>
                    <version>1.10.0</version>
                </dependency>
                <dependency>
                    <groupId>org.apache.flink</groupId>
                    <artifactId>flink-csv</artifactId>
                    <version>1.10.0</version>
                </dependency>

                <dependency>
                    <groupId>org.apache.flink</groupId>
                    <artifactId>flink-table-planner_${scala.binary.version}</artifactId
>
                    <version>${flink.version}</version>
                    <scope>provided</scope>
                </dependency>

                <dependency>
                    <groupId>org.apache.flink</groupId>
                    <artifactId>flink-jdbc_2.11</artifactId>
                    <version>${flink.version}</version>
                    <scope>provided</scope>
                </dependency>

                <!-- CLI dependencies -->
                <dependency>
                    <groupId>org.apache.flink</groupId>
                    <artifactId>flink-clients_2.11</artifactId>
                    <version>${flink.version}</version>
                    <scope>provided</scope>
                    <exclusions>
                        <exclusion>
                            <artifactId>javassist</artifactId>
                            <groupId>org.javassist</groupId>
                        </exclusion>
                        <exclusion>
                            <artifactId>scala-parser-combinators_2.11</artifactId>
                            <groupId>org.scala-lang.modules</groupId>
                        </exclusion>
```

```
                        <exclusion>
                            <artifactId>slf4j-api</artifactId>
                            <groupId>org.slf4j</groupId>
                        </exclusion>
                        <exclusion>
                            <artifactId>snappy-java</artifactId>
                            <groupId>org.xerial.snappy</groupId>
                        </exclusion>
                    </exclusions>
                </dependency>

                <dependency>
                    <groupId>org.apache.flink</groupId>
                    <artifactId>flink-java</artifactId>
                    <version>${flink.version}</version>
                    <scope>provided</scope>
                </dependency>

                <dependency>
                    <groupId>org.apache.flink</groupId>
                    <artifactId>flink-streaming-java_${scala.binary.version}</artifactI
d>
                    <version>${flink.version}</version>
                    <scope>provided</scope>
                </dependency>
                <!-- https://mvnrepository.com/artifact/org.apache.kafka/kafka-clients
-->
                <dependency>
                    <groupId>org.apache.kafka</groupId>
                    <artifactId>kafka-clients</artifactId>
                    <version>0.11.0.3</version>
                    <exclusions>
                        <exclusion>
                            <artifactId>slf4j-api</artifactId>
                            <groupId>org.slf4j</groupId>
                        </exclusion>
                    </exclusions>
                </dependency>

                <!-- https://mvnrepository.com/artifact/org.apache.flink/flink-hbase --
>
                <dependency>
                    <groupId>org.apache.flink</groupId>
                    <artifactId>flink-hbase_2.11</artifactId>
                    <version>1.10.0</version>
                </dependency>
                <dependency>
                    <groupId>org.apache.hadoop</groupId>
                    <artifactId>hadoop-common</artifactId>
                    <version>2.7.0</version>
                </dependency>

                <dependency>
                    <groupId>org.apache.flink</groupId>
                    <artifactId>flink-connector-kafka-0.11_${scala.binary.version}</art
ifactId>
                    <version>${flink.version}</version>
                    <exclusions>
                        <exclusion>
                            <artifactId>kafka-clients</artifactId>
                            <groupId>org.apache.kafka</groupId>
                        </exclusion>
                    </exclusions>
```

```
            </dependency>

            <!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
            <dependency>
                <groupId>mysql</groupId>
                <artifactId>mysql-connector-java</artifactId>
                <version>5.1.37</version>
            </dependency>

            <dependency>
                <groupId>com.fasterxml.jackson.core</groupId>
                <artifactId>jackson-core</artifactId>
                <version>2.9.5</version>
            </dependency>

            <dependency>
                <groupId>io.lettuce</groupId>
                <artifactId>lettuce-core</artifactId>
                <version>5.1.8.RELEASE</version>
            </dependency>

            <dependency>
                <groupId>com.alibaba</groupId>
                <artifactId>fastjson</artifactId>
                <version>1.2.46</version>
            </dependency>


            <dependency>
                <groupId>org.apache.flink</groupId>
                <artifactId>flink-table-api-java-bridge_2.11</artifactId>
                <version>1.10.0</version>
                <scope>provided</scope>
            </dependency>

        <!-- <dependency>-->
        <!-- <groupId>io.netty</groupId>-->
        <!-- <artifactId>netty-all</artifactId>-->
        <!-- <version>4.1.4.Final</version>-->
        <!-- </dependency>-->

            <!-- https://mvnrepository.com/artifact/org.apache.flink/flink-jdbc -->
            <dependency>
                <groupId>org.apache.flink</groupId>
                <artifactId>flink-jdbc_2.11</artifactId>
                <version>1.10.0</version>
            </dependency>

        </dependencies>
        <build>
            <plugins>
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-compiler-plugin</artifactId>
                    <version>3.8.1</version>
                    <configuration>
                        <encoding>UTF-8</encoding>
                        <source>1.8</source>
                        <target>1.8</target>
                    </configuration>
                </plugin>
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
```

```
                    <artifactId>maven-shade-plugin</artifactId>
                    <version>2.4.3</version>
                    <executions>
                        <execution>
                            <phase>package</phase>
                            <goals>
                                <goal>shade</goal>
                            </goals>
                            <configuration>
                                <filters>
                                    <filter>
                                        <artifact>*:*</artifact>
                                        <excludes>
                                            <exclude>META-INF/*.SF</exclude>
                                            <exclude>META-INF/*.DSA</exclude>
                                            <exclude>META-INF/*.RSA</exclude>
                                        </excludes>
                                    </filter>
                                </filters>
                                <artifactSet>
                                    <excludes>
                                        <exclude>junit:junit</exclude>
                                    </excludes>
                                </artifactSet>

                            </configuration>
                        </execution>
                    </executions>
                </plugin>
            </plugins>
        </build>
    </project>

    ```
```
```