基于Flink 1.10 的SQL CONNECTOR开发

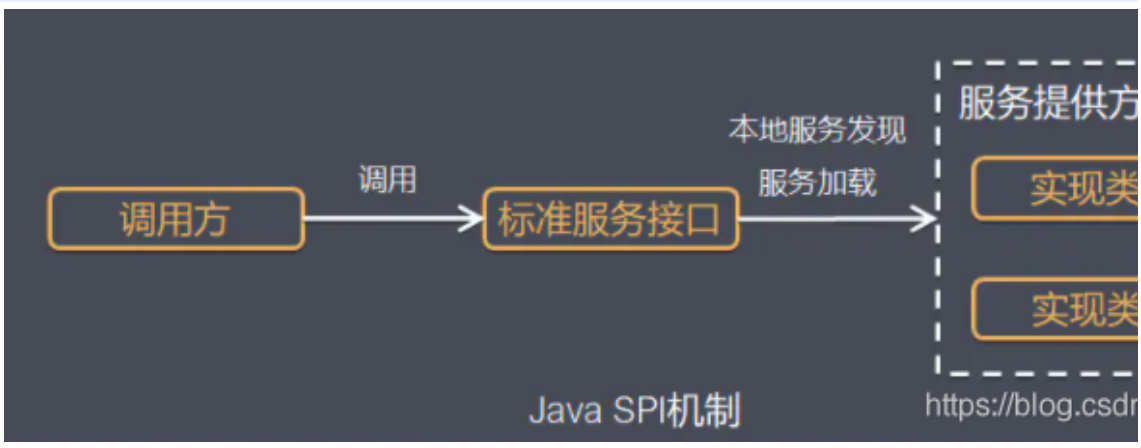
2020-06-22 | technology

Flink 1.10提供了丰富的connector组件帮助用户连接外部系统。但是很多时候原生的connector并不能够完全满足用户的需求，因此需要自定义开发connector组件。本文介绍如何进行Flink1.10 SQL CONNECTOR的开发工作。

Flink 1.10通过SPI去加载不同的factory，实现了CONNECTOR的统一。

SPI机制

SPI，全称为Service Provider Interface，是Java提供的一套用于第三方实现或拓展的API。



SPI原理

基于工厂模式的任务提交

```
public interface PipelineExecutor{
                /**
                 * 执行任务
                 */
                CompletableFuture<JobClient> execute(final Pipeline pipeline, f
}
```

PipelineExecutor **提供多种实现方式：**

RemoteExecutor(standalone)

LocalExecutor (local)

YanrJobClusterExecutor (per-job)

YarnSessionClusterExecutor (yarn-session)

支持用户在多种场景下提交flink任务。

Flink 提供PipelineExecutorServiceLoader接口，其中实现类DefaultExecutorServiceLoader支持通过名称去加载类。它的原理就是通过SPI机制去查找flink所提供的所有的工厂类，找到合适的类，进行加载。

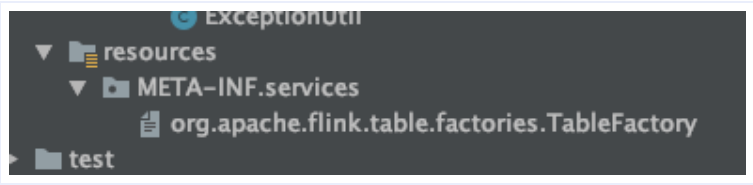
基于工厂模式的SQL CONNECTOR设计

Flink 1.10提供SPI方式支持与connector进行交互，Flink会去扫描包中resources/META-INF/services目录下的org.apache.flink.table.factories.TableFactory，获取所有Factory类，根据sql中with传进来的参数（k=v）进行匹配，找到匹配到的那个Factory类，如果没有找到的话，则会报错。

```
# Licensed to the Apache Software Foundation (ASF) under one or mor
# contributor license agreements.  See the NOTICE file distributed
# this work for additional information regarding copyright ownershi
# The ASF licenses this file to You under the Apache License, Versi
# (the "License"); you may not use this file except in compliance w
# the License.  You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, softwa
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or i
# See the License for the specific language governing permissions a
# limitations under the License.

[redacted]flink.streaming.connectors.elasticsearch7.ElasticsearchT
```

配置工厂类的内容



配置工厂类的文件

```java
public class SqlHdfsSinkFactory implements StreamTableSinkFactory<Row> {

    @Override
    public List<String> supportedProperties() {
        List<String> properties = new ArrayList<>();
        properties.add(SqlConstants.HIVE_CONNECTOR_PROPERTY_DB_NAME);
        properties.add(SqlConstants.HIVE_CONNECTOR_PROPERTY_TABLE_NAME);
        properties.add(SqlConstants.HIVE_CONNECTOR_PROPERTY_CONFIG);

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

        return properties;
    }

```

```
    @Override
    public Map<String, String> requiredContext() {
        Map<String, String> context = new HashMap<>();
        context.put(SqlConstants.HIVE_CONNECTOR_PROPERTY_CONNECTOR_TYPE, SqlCo
        context.put(SqlConstants.HIVE_CONNECTOR_PROPERTY_PACKAGE_VERSION, hive
        context.put(CONNECTOR_PROPERTY_VERSION, "1"); // backwards compatibili

        return context;
    }

    @Override
    public TableSink createTableSink(Map properties) {
        return new SqlHdfsSink(properties
                .get(SqlConstants.HIVE_CONNECTOR_PROPERTY_CONFIG).toString());
    }

    @Override
    public StreamTableSink createStreamTableSink(Map properties) {

        return new SqlHdfsSink(properties
                .get(SqlConstants.HIVE_CONNECTOR_PROPERTY_CONFIG).toString());

    }

    private String hiveVersion() {
        return SqlConstants.CONNECTOR_HIVE_VERSION_VALUE_211;
    }
}
```

其中对几个方法进行了重写：

supportedProperties() 记录sink参数对应的k-v值，这里参数值是固定的

requiredContext() 记录sink参数对应的k-v值，这里参数值是支持动态的

createTableSink() 创建sink对象的入口, 其中properties参数表示sql with语句中的k-v。

在createTableSink()方法中，实例化了SqlHdfsSink类，这个类实现了AppendStreamTableSink接口，是真正将数据写入到HDFS中的核心类。

```
public class SqlHdfsSink implements AppendStreamTableSink<Row> { }
```

使用自定义的SQL SINK

Flink 1.10使用classLoader方式加载sql sink，对于用户来讲使用起来比较简单，定义好`StreamTableEnvironment stEnv = StreamTableEnvironment.create(env, settings)`即可将SQL语句传到stEnv中，自动加载sink。

```
stEnv.sqlUpdate("CREATE TABLE gis_log_sink (\n" +
                "    deptSiteId STRING,\n" +
                "    latitude STRING,\n" +
```

```
"   passZoneCode STRING\n" +
"   ) \n" +
"with (\n" +
"    'version' = '2.1.1',\n" +
"    'dbName' = 'dm_drcs',\n" +
"    'connectorType' = 'sink',\n" +
"    'connectorName' = 'HIVE',\n" +
"    'columnDelimiter' = '\\u0001',\n" +
"    'hadoopClusterName' = 'bdp_sit',\n" +
"    'timeoutRollInterval' = '1000',\n" +
"    'configuration' = '{\"properties\":{\"batchSize\":1048576
")");
```

总结

Flink 1.10基于SPI机制加载CONNECTOR ， 统一了Flink 与外部系统的交互，也降低了用户实现自定义CONNECTOR的门槛。

开发自定义SQL CONNECTOR，本质上是在原有的SINK类前面封装一个factory类，Flink会去读取这个factory类，并将其加载到runtime中执行。

感谢阅读!
祝大家早安，午安，晚安.....