Flink Redis Sink用法与支持过期时间的改进

Flink默认提供了很多开箱即用的连接器，比如与Kafka、RabbitMQ、HDFS、ElasticSearch等对接的连接器。还有一些不那么常用的连接器则由Apache Bahir项目（官网很简陋，见这里）来提供，其中就包含Redis Sink。这个项目的文档有点缺乏，本文先记录一下用法。

引入如下Maven依赖。目前bahir-flink项目比较停滞，最新版本是1.1-SNAPSHOT。

```
    <dependency>
      <groupId>org.apache.bahir</groupId>
      <artifactId>flink-connector-redis_${scala.bin.version}</artifactId>
      <version>${bahir.version}</version>
      <scope>compile</scope>
    </dependency>
```

以最常见的单机Redis情景来讨论，该插件提供的核心类有三个，分别是：

- FlinkJedisPoolConfig类：Jedis连接池的相关参数；
- RedisMapper接口：从用户数据中提取键值，并构成Redis命令的映射器，需要用户自己实现；
- RedisSink类：根据构建好的FlinkJedisPoolConfig和RedisMapper将流数据写入Redis。

先生成一个FlinkJedisPoolConfig实例。

```
    // 这个叫ParameterUtil的类是自己写的，专门用来读有占位符的属性文件，看官勿误会
    // 当然也可以直接写明文，不过放在属性文件里方便管理，还能按Maven profile作区分

  Properties redisProps = ParameterUtil.getFromResourceFile("redis.properties
    |
  FlinkJedisPoolConfig jedisPoolConfig = new FlinkJedisPoolConfig.Builder()
      .setHost(redisProps.getProperty("redis.host"))
    .setPort(NumberUtils.createInteger(redisProps.getProperty("redis.port")))
      .setPassword(redisProps.getProperty("redis.pass", ""))
    .setDatabase(NumberUtils.createInteger(redisProps.getProperty("redis.db"
      .build();
```

接下来就写一个RedisMapper的实现类，它负责将窗口统计出来的PV和UV数据以JSON形式表示。一点都不难。

```
public static final class RedisWindowPvUvMapper
  implements RedisMapper<WindowedPvUvResult> {
  // 被统计的对象类别，当参数传进来
  private String itemType;

  public RedisStringMapper(String itemType) {
    this.itemType = itemType;
  }

  // 指定命令，这里要写字符串，所以是set
  @Override
  public RedisCommandDescription getCommandDescription() {
    return new RedisCommandDescription(RedisCommand.SET);
  }

  // 从POJO构造key
  @Override
  public String getKeyFromData(WindowedPvUvResult result) {
    StringBuilder builder = new StringBuilder("flink:log_pvuv:");
    builder.append(result.getWindowEnd());
    builder.append("_");
    builder.append(itemType);
    builder.append("_");
    builder.append(result.getItemId());
    return builder.toString();
  }

  // 从POJO构造value
  @Override
  public String getValueFromData(WindowedPvUvResult result) {
    return new JSONObject()
      .fluentPut("pv", result.getPv())
      .fluentPut("uv", result.getUv())
      .toJSONString();
  }
}
```

最后就可以构造RedisSink了。

```
dataStream.addSink(new RedisSink<>(jedisPoolConfig, new RedisWindowPvUvMapper("
partner")));
```

这个Redis连接器简单易用，但是有两个地方差强人意：一是不支持设定key的过期时间（TTL），二是不支持流水线（pipeline）。在窗口比较稀疏、写入量没那么大的情况下，流水线是可有可无的，但过期时间还是很重要，所以下面要稍微改造一下。

将项目代码clone到本地，找到flink-connector-redis项目中的RedisCommand枚举，加上setex命令。

```
SETEX(RedisDataType.STRING),
```

然后来到RedisCommandsContainer接口，它其中定义的都是具体的命令逻辑，加上setex()方法的定义。

```
void setex(String key, int seconds, String value);
```

RedisCommandsContainer接口有两个实现类：针对单机的RedisContainer和针对集群的RedisClusterContainer，写入setex()方法的具体实现。

```
    // RedisContainer.setex()
    @Override

  public void setex(final String key, final int seconds, final String value)
        Jedis jedis = null;          try {
            jedis = getInstance();
            jedis.setex(key, seconds, value);
        } catch (Exception e) {
            if (LOG.isErrorEnabled()) {

             LOG.error("Cannot send Redis message with command SETEX to key
                  key, e.getMessage());              }
            throw e;
        } finally {
            releaseInstance(jedis);
        }
    }

    // RedisClusterContainer.setex()
    @Override

  public void setex(final String key, final int seconds, final String value)
        try {              jedisCluster.setex(key, seconds, value);
        } catch (Exception e) {
            if (LOG.isErrorEnabled()) {

             LOG.error("Cannot send Redis message with command SETEX to key
```

```
                    key, e.getMessage());
            throw e;
        }
    }
```

有了方法的具体实现，那么如何接收TTL的参数呢？回到上面提到过的RedisMapper接口，在其中加上一个获取TTL秒数的方法声明。为了方便，还可以用default语法提供一个默认值。

```
    default int getExpireSeconds(T data) {
        return 0;
    }
```

万事俱备只欠东风，来到RedisSink.invoke()方法（之前已经讲过，RichSinkFunction的子类必须实现这个方法），加上我们之前写的东西就可以了，如下。

```
    @Override
    public void invoke(IN input, Context context) throws Exception {
        String key = redisSinkMapper.getKeyFromData(input);
        String value = redisSinkMapper.getValueFromData(input);
        // 取得过期时间
        int expireSec = redisSinkMapper.getExpireSeconds(input);

        Optional<String> optAdditionalKey = redisSinkMapper.getAdditionalKey(i
            switch (redisCommand) {
            case RPUSH:
                this.redisCommandsContainer.rpush(key, value);
                break;
            case LPUSH:
                this.redisCommandsContainer.lpush(key, value);
                break;
            case SADD:
                this.redisCommandsContainer.sadd(key, value);
                break;
            case SET:
                this.redisCommandsContainer.set(key, value);
                break;
            // 新写的setex逻辑
            case SETEX:
                if (expireSec > 0) {

                    this.redisCommandsContainer.setex(key, expireSec, value);
                }                   break;
            case PFADD:
            // ...以下原样，略去
        }
```

```
    }
```

用Maven重新构建、打包并发布到仓库就可以用了。在实际应用时，如果需要设定TTL，用户逻辑中的RedisMapper就可以这样写：

```
  public static final class RedisStringMapperWithTTL
    implements RedisMapper<WindowedPvUvResult> {
    @Override
    public RedisCommandDescription getCommandDescription() {
      return new RedisCommandDescription(RedisCommand.SETEX);
    }

    @Override
    public String getKeyFromData(WindowedPvUvResult result) {
      // ...
    }

    @Override
    public String getValueFromData(WindowedPvUvResult result) {
      // ...
    }

    @Override
    public int getExpireSeconds(WindowedPvUvResult data) {
      return 3 * 24 * 60 * 60;   // 3天
    }
  }
}
```

so easy~