



[Home](#) ([//ci.apache.org/projects/flink/flink-docs-release-1.10/zh/](https://ci.apache.org/projects/flink/flink-docs-release-1.10/zh/))

## Getting Started


[概览](#) ([//ci.apache.org/projects/flink/flink-docs-release-1.10/zh/getting-started/](https://ci.apache.org/projects/flink/flink-docs-release-1.10/zh/getting-started/))


### Code Walkthroughs

[DataStream API](#) ([//ci.apache.org/projects/flink/flink-docs-release-1.10/zh/getting-started/walkthroughs/datastream\\_api.html](https://ci.apache.org/projects/flink/flink-docs-release-1.10/zh/getting-started/walkthroughs/datastream_api.html))


[Table API](#) ([//ci.apache.org/projects/flink/flink-docs-release-1.10/zh/getting-started/walkthroughs/table\\_api.html](https://ci.apache.org/projects/flink/flink-docs-release-1.10/zh/getting-started/walkthroughs/table_api.html))


 [Docker Playgrounds](#)


 [教程](#)

 [示例](#)


 [概念](#)

 [应用开发](#)


 [部署与运维](#)

 [调试和监控](#)

 [Flink 开发](#)

 [内幕](#)

 [Javadocs](#) ([//ci.apache.org/projects/flink/flink-docs-release-1.10/api/java](https://ci.apache.org/projects/flink/flink-docs-release-1.10/api/java))

 [Scaladocs](#) ([//ci.apache.org/projects/flink/flink-docs-release-1.10/api/scala/index.html#org.apache.flink.api.scala.package](https://ci.apache.org/projects/flink/flink-docs-release-1.10/api/scala/index.html#org.apache.flink.api.scala.package))

 [Pythondocs](#) ([//ci.apache.org/projects/flink/flink-docs-release-1.10/api/python](https://ci.apache.org/projects/flink/flink-docs-release-1.10/api/python))

 [Project Page](#) (<http://flink.apache.org>)

# Table API

Apache Flink 提供 Table API 作为批处理和流处理统一的关系型API，即查询在无界实时流或有界批数据集上以相同的语义执行，并产生相同的结果。Flink 中的 Table API 通常用于简化数据分析，数据流水线 和 ETL 应用程序的定义。

接下来你会构建什么？
先决条件
救命，我被困住了！
如何跟进
代码详解
尝试一下
添加窗口
通过流处理的方式再来一次！
最终程序

## 接下来你会构建什么？

在本教程中，你将学习如何构建连续的 ETL 流水线，以便按账户随时跟踪金融交易。首先你将报表构建为每晚执行的批处理作业，然后迁移到流式管道。

# 先决条件

本演练假设你对 Java 和 Scala 有一定的了解，但即便你使用其他编程语言，相信也可以学会。它还假定你熟悉基本的关系概念比如 SELECT 和 GROUP BY 子句。

# 救命，我被困住了！

如果你被难题困住了，可以在社区 (<https://flink.apache.org/community.html>)寻求帮助。值得一提的是，Apache Flink 的用户邮件列表 (<https://flink.apache.org/community.html#mailing-lists>)一直是最活跃的 Apache 项目之一，也是一个快速获得帮助的好途径。

# 如何跟进

如果想要继续，你的电脑需要安装：

- Java 8 or 11
- Maven

现成的 Flink Maven Archetype 可以快速创建一个具有所有必要依赖的框架项目：

Java	Scala
------	-------

```
$ mvn archetype:generate \
  -DarchetypeGroupId=org.apache.flink \
  -DarchetypeArtifactId=flink-walkthrough-table-java \
  -DarchetypeVersion=1.10.0 \
  -DgroupId=spend-report \
  -DartifactId=spend-report \
  -Dversion=0.1 \
  -Dpackage=spendreport \
  -DinteractiveMode=false
```

你可以根据自己的意愿修改 groupId、artifactId 和 package 参数。通过使用以上参数，Maven 将创建一个拥有全部所需依赖的项目来完成本教程。把项目导入编辑器后，你会看到一个包含以下代码的文件，你可以在 IDE 中直接运行它。

Java	Scala
------	-------

```
ExecutionEnvironment env = ExecutionEnvironment.getExecutionEnvironment();
BatchTableEnvironment tEnv = BatchTableEnvironment.create(env);

tEnv.registerTableSource("transactions", new BoundedTransactionTableSource());
tEnv.registerTableSink("spend_report", new SpendReportTableSink());
tEnv.registerFunction("truncateDateToHour", new TruncateDateToHour());

tEnv
    .scan("transactions")
    .insertInto("spend_report");

env.execute("Spend Report");
```

# 代码详解

## 运行环境

前两行设置了你的 `ExecutionEnvironment`。运行环境用来设置作业的属性、指定应用是批处理还是流处理，以及创建数据源。由于你正在建立一个定时的批处理报告，本教程以批处理环境作为开始。然后将其包装进 `BatchTableEnvironment` 中从而能够使用所有的 `Tabel API`。

Java	Scala
------	-------

```
ExecutionEnvironment env = ExecutionEnvironment.getExecutionEnvironment();
BatchTableEnvironment tEnv = BatchTableEnvironment.create(env);
```

## 注册表

接下来，表将会被注册到运行环境之中，这样你就可以用它们连接外部系统以读取或写入批数据或流数据。`source` 提供对存储在外部系统中的数据的访问；例如数据库、键-值存储、消息队列或文件系统。`sink` 则将表中的数据发送到外部存储系统。根据 `source` 或 `sink` 的类型，它们支持不同的格式，如 `CSV`、`JSON`、`Avro` 或 `Parquet`。

Java	Scala
------	-------

```
tEnv.registerTableSource("transactions", new BoundedTransactionTableSource());
tEnv.registerTableSink("spend_report", new SpendReportTableSink());
```

上例代码注册了两张表。交易表作为输入表，支出报告表作为输出表。我们可以从交易（`transactions`）表中读取信用卡的交易记录，其中包含了账户 ID（`accountId`）字段、时间戳（`timestamp`）字段和交易金额（`amount`）字段。本教程中，该表使用内存中的数据，以避免对外部系统的任何依赖。而在实际情况下，

BoundedTransactionTableSource 可能来源于文件系统、数据库或任何静态数据源。支出报告表 `spend_report` 用 **INFO** 日志级别将表的每一行数据记录到日志，而不是写入持久化存储，所以你可以很容易地查看结果。

## 注册 UDF

一个用来处理时间戳的自定义函数 ([//ci.apache.org/projects/flink/flink-docs-release-1.10/zh/dev/table/functions/udfs.html](https://ci.apache.org/projects/flink/flink-docs-release-1.10/zh/dev/table/functions/udfs.html))随表一起被注册到tEnv中。此函数将时间戳向下舍入到最近的小时。

Java	Scala
------	-------

```
tEnv.registerFunction("truncateDateToHour", new TruncateDateToHour());
```

## 查询

完成配置环境和注册表后，你已准备好构建第一个应用程序。从 `TableEnvironment` 中，你可以 `scan` 一个输入表读取其中的行，然后用 `insertInto` 把这些数据写到输出表中。

Java	Scala
------	-------

```
tEnv
    .scan("transactions")
    .insertInto("spend_report");
```

最初，作业读取所有的交易记录并用 **INFO** 日志级别将其记录下来。

## 运行

Flink 应用是延迟构建的，只有完全定义好之后才交付集群运行。你可以调用 `ExecutionEnvironment#execute` 来开始作业的执行并给它取一个名字。

Java	Scala
------	-------

```
env.execute("Spend Report");
```

# 尝试一下

现在有了作业设置的框架，你就可以添加一些业务逻辑了。目标是建立一个报表来显示每天每小时每个账户的总支出。就像一个 SQL 查询一样，Flink 可以选取所需的字段并且按键分组。由于时间戳字段具有毫秒的粒度，你可以使用自定义函数将其舍入到最近的小时。最后，选取所有的字段，用内建的 `sum` 聚合函数

(<https://ci.apache.org/projects/flink/flink-docs-release-1.10/zh/dev/table/functions/systemFunctions.html#aggregate-functions>)函数合计每一个账户每小时的支出。

Java	Scala
------	-------

```
tEnv
  .scan("transactions")
  .select("accountId, timestamp.truncateDateToHour as timestamp, amount")
  .groupBy("accountId, timestamp")
  .select("accountId, timestamp, amount.sum as total")
  .insertInto("spend_report");
```

这个查询处理了 transactions 表中的所有记录，计算报告，并以高效、可扩展的方式输出结果。

# 查询 1 的输出显示了账户 id、时间戳和消费总额。

```
> 1, 2019-01-01 00:00:00.0, $567.87
> 2, 2019-01-01 00:00:00.0, $726.23
> 1, 2019-01-01 01:00:00.0, $686.87
> 2, 2019-01-01 01:00:00.0, $810.06
> 1, 2019-01-01 02:00:00.0, $859.35
> 2, 2019-01-01 02:00:00.0, $458.40
> 1, 2019-01-01 03:00:00.0, $330.85
> 2, 2019-01-01 03:00:00.0, $730.02
> 1, 2019-01-01 04:00:00.0, $585.16
> 2, 2019-01-01 04:00:00.0, $760.76
```

## 添加窗口

根据时间进行分组在数据处理中是一种很常见的方式，特别是在处理无限的数据流时。基于时间的分组称为窗口 (<https://ci.apache.org/projects/flink/flink-docs-release-1.10/zh/dev/stream/operators/windows.html>)，Flink 提供了灵活的窗口语义。其中最基础的是 Tumble window（滚动窗口），它具有固定大小且窗口之间不重叠。

Java	Scala
------	-------

```
tEnv
  .scan("transactions")
  .window(Tumble.over("1.hour").on("timestamp").as("w"))
  .groupBy("accountId, w")
  .select("accountId, w.start as timestamp, amount.sum")
  .insertInto("spend_report");
```

你的应用将会使用基于时间戳字段的一小时的滚动窗口。因此时间戳是 2019-06-01 01:23:47 的行被放入 2019-06-01 01:00:00 这个时间窗口之中。

在持续的流式应用中，基于时间的聚合结果是唯一的，因为相较于其他属性，时间通常会向前移动。在批处理环境中，窗口提供了一个方便的 API，用于按时间戳属性对记录进行分组。

运行这个更新过的查询将会得到和之前一样的结果。

```
# 查询 2 的输出显示了账户 id、时间戳和消费总额

> 1, 2019-01-01 00:00:00.0, $567.87
> 2, 2019-01-01 00:00:00.0, $726.23
> 1, 2019-01-01 01:00:00.0, $686.87
> 2, 2019-01-01 01:00:00.0, $810.06
> 1, 2019-01-01 02:00:00.0, $859.35
> 2, 2019-01-01 02:00:00.0, $458.40
> 1, 2019-01-01 03:00:00.0, $330.85
> 2, 2019-01-01 03:00:00.0, $730.02
> 1, 2019-01-01 04:00:00.0, $585.16
> 2, 2019-01-01 04:00:00.0, $760.76
```

# 通过流处理的方式再来一次！

因为 Flink 的 Table API 为批处理和流处理提供了相同的语法和语义，从一种方式迁移到另一种方式只需要两步。

第一步是把批处理的 `ExecutionEnvironment` 替换成流处理对应的 `StreamExecutionEnvironment`，后者创建连续的流作业。它包含特定于流处理的配置，比如时间特性。当这个属性被设置成 事件时间 ([//ci.apache.org/projects/flink/flink-docs-release-1.10/zh/dev/event\\_time.html](https://ci.apache.org/projects/flink/flink-docs-release-1.10/zh/dev/event_time.html))时，它能保证即使遭遇乱序事件或者作业失败的情况也能输出一致的结果。滚动窗口在对数据进行分组时就运用了这个特性。

Java	Scala
------	-------

```
StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
env.setStreamTimeCharacteristic(TimeCharacteristic.EventTime);
```

```
StreamTableEnvironment tEnv = StreamTableEnvironment.create(env);
```

第二步就是把有界的数据源替换成无限的数据源。这个项目通过 `UnboundedTransactionTableSource` 持续不断地实时生成交易事件。与 `BoundedTransactionTableSource` 一样，这个表也是通过在内存中生成数据从而不依赖外部系统。在实践中，这个表可能从一个流式数据源中读取数据，比如 Apache Kafka、AWS Kinesis 或者 Pravega。

Java	Scala
------	-------

```
tEnv.registerTableSource("transactions", new UnboundedTransactionTableSource());
```

这就是一个功能齐全、有状态的分布式流式应用！ 这个查询会持续处理交易流，计算每小时的消费额，然后实时输出结果。由于输入是无界的，因此查询将一直运行，直到手动停止为止。因为这个作业使用了基于时间窗口的聚合，Flink 可以使用一些特定的优化，比如当系统知道一个特定的窗口不会再有新的数据到来，它就会对状态进行清理。

```
# 查询 3 的输出显示了账户 id、时间戳消费总额

> 1, 2019-01-01 00:00:00.0, $567.87
> 2, 2019-01-01 00:00:00.0, $726.23

# 这些行是在这一小时中连续计算的
# 并在这一小时结束时立刻输出

> 1, 2019-01-01 01:00:00.0, $686.87
> 2, 2019-01-01 01:00:00.0, $810.06

# 当接收到该窗口的第一条数据时
# Flink 就开始计算了
```

# 最终程序

Java	Scala
------	-------



```

package spendreport;

import org.apache.flink.walkthrough.common.table.SpendReportTableSink;
import org.apache.flink.walkthrough.common.table.TransactionTableSource;
import org.apache.flink.streaming.api.TimeCharacteristic;
import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
import org.apache.flink.table.api.Tumble;
import org.apache.flink.table.api.java.StreamTableEnvironment;

public class SpendReport {

    public static void main(String[] args) throws Exception {
        StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
        env.setStreamTimeCharacteristic(TimeCharacteristic.EventTime);

        StreamTableEnvironment tEnv = StreamTableEnvironment.create(env);

        tEnv.registerTableSource("transactions", new UnboundedTransactionTableSource());
        tEnv.registerTableSink("spend_report", new SpendReportTableSink());

        tEnv
            .scan("transactions")
            .window(Tumble.over("1.hour").on("timestamp").as("w"))
            .groupBy("accountId, w")
            .select("accountId, w.start as timestamp, amount.sum")
            .insertInto("spend_report");

        env.execute("Spend Report");
    }
}

```

---

想参与贡献翻译? (<https://cwiki.apache.org/confluence/display/FLINK/Flink+Translation+Specifications>)