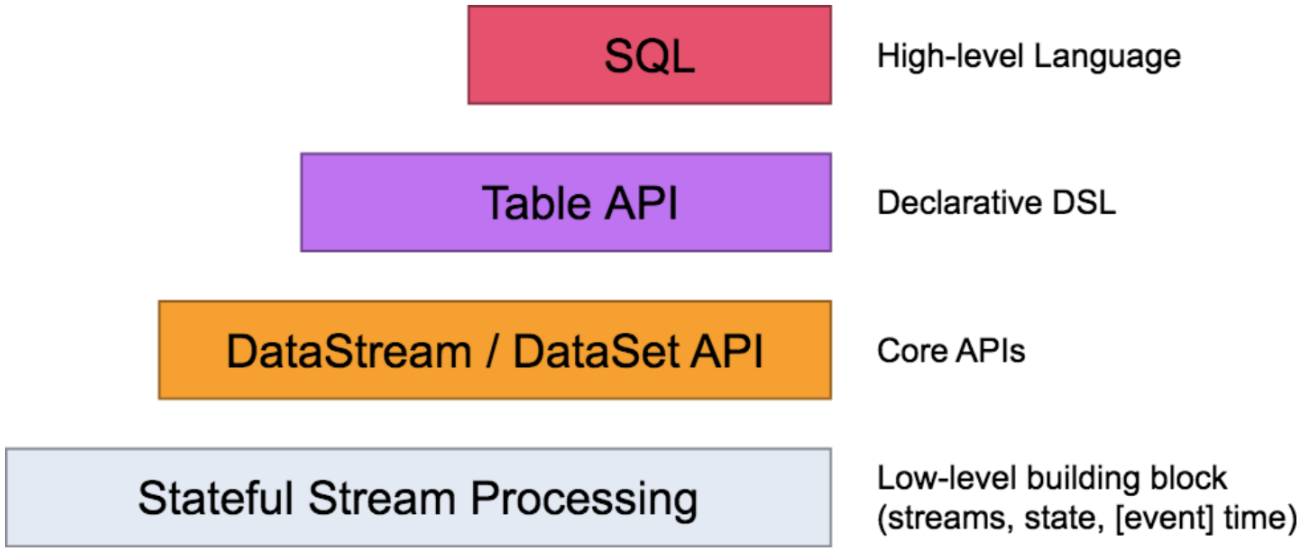
flink系列-8、Flink Table API & Flink Sql API

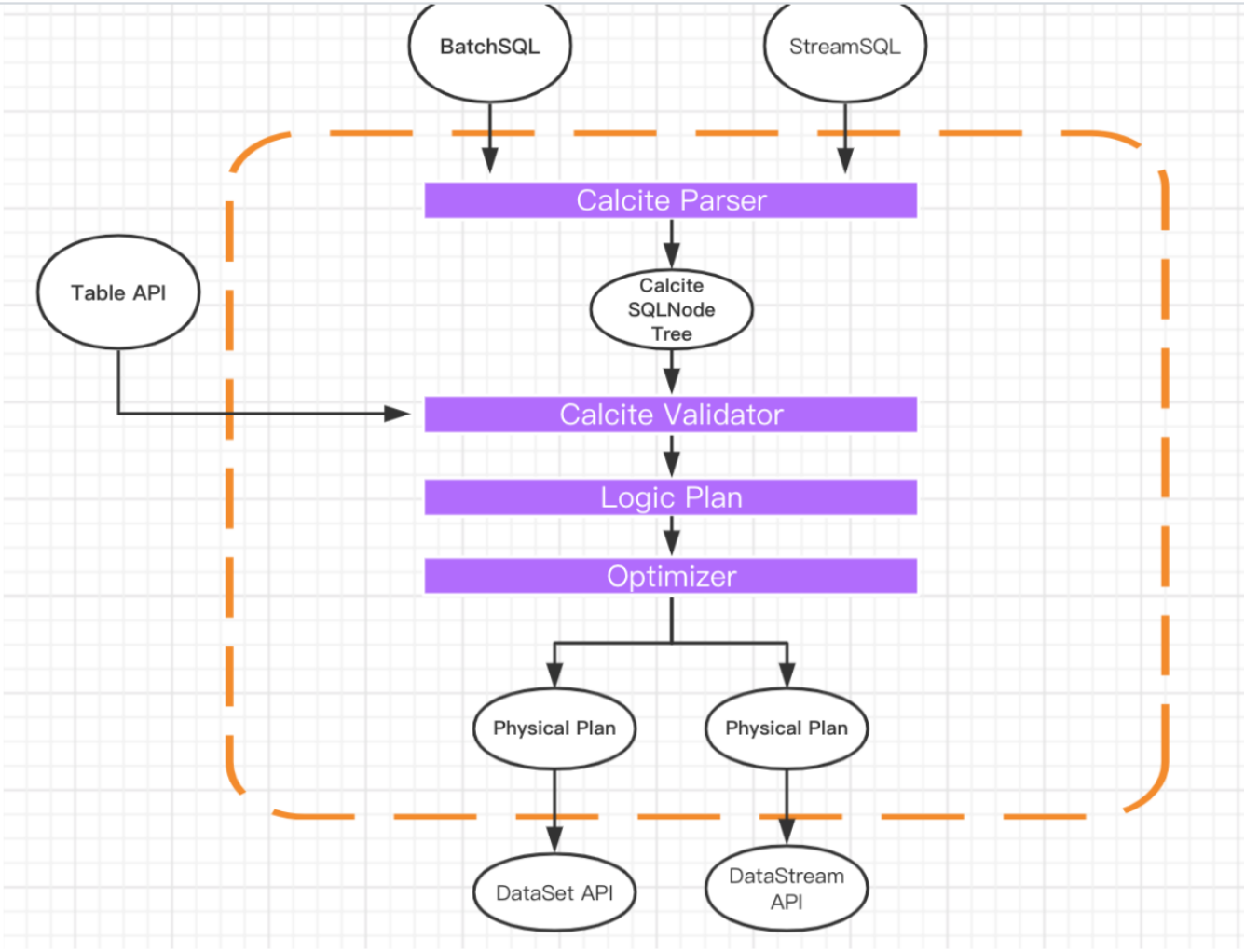
一、概述



- 上图是flink的分层模型，Table API 和 SQL 处于最顶端，是 Flink 提供的高级 API 操作。Flink SQL 是 Flink 实时计算为简化计算模型，降低用户使用实时计算门槛而设计的一套符合标准 SQL 语义的开发语言。
- Flink 在编程模型上提供了 DataStream 和 DataSet 两套 API，并没有做到事实上的批流统一，因为用户和开发者还是开发了两套代码。正是因为 Flink Table & SQL 的加入，可以说 Flink 在某种程度上做到了事实上的批流一体。

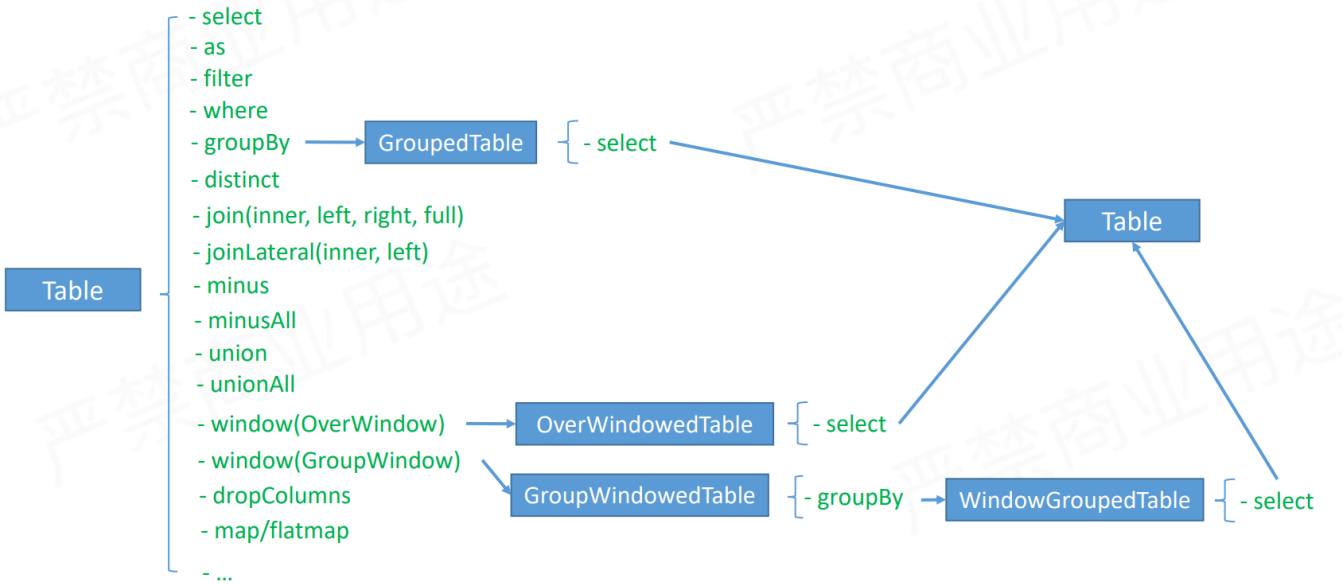
原理

- Flink 底层对 SQL 的解析，优化，执行用到了 Apache Calcite。
- 下图是一张经典的 Flink Table & SQL 实现原理图，可以看到 Calcite 在整个架构中处于绝对核心地位。

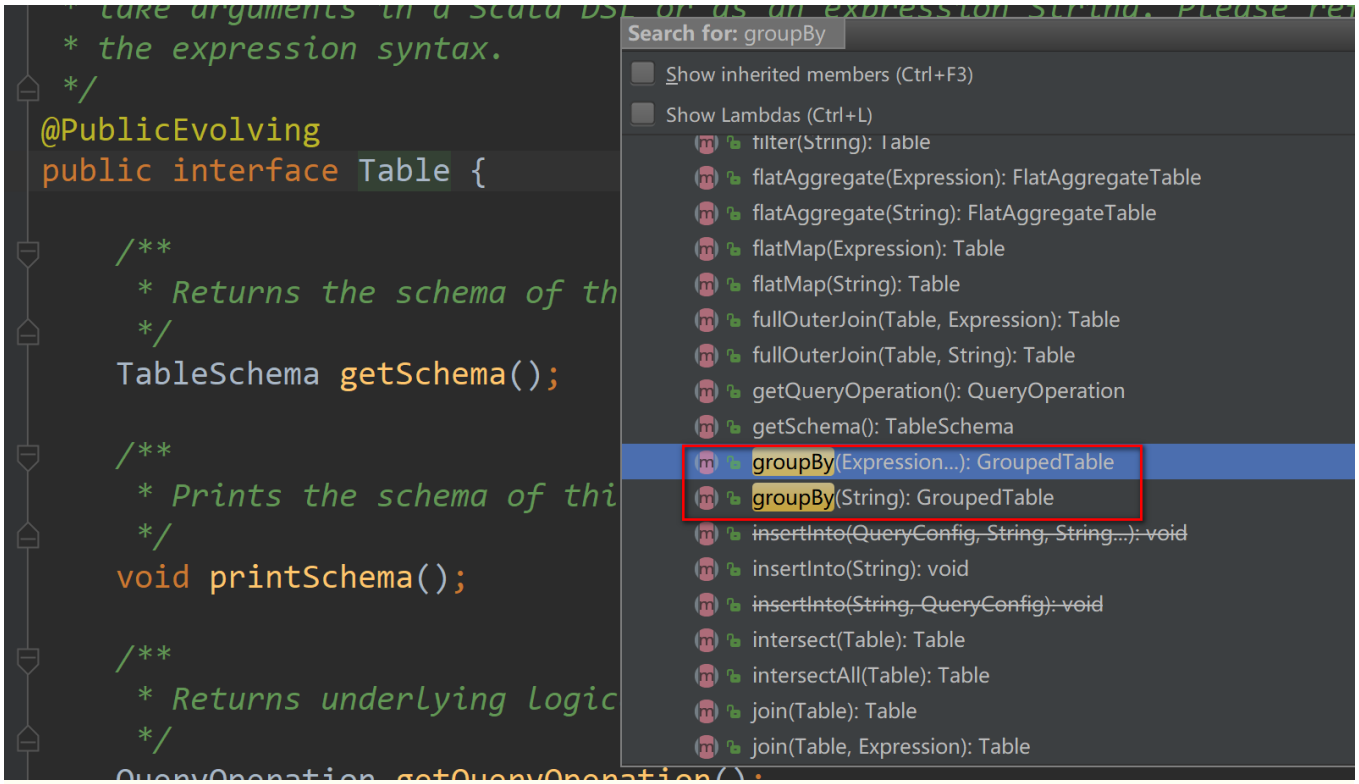


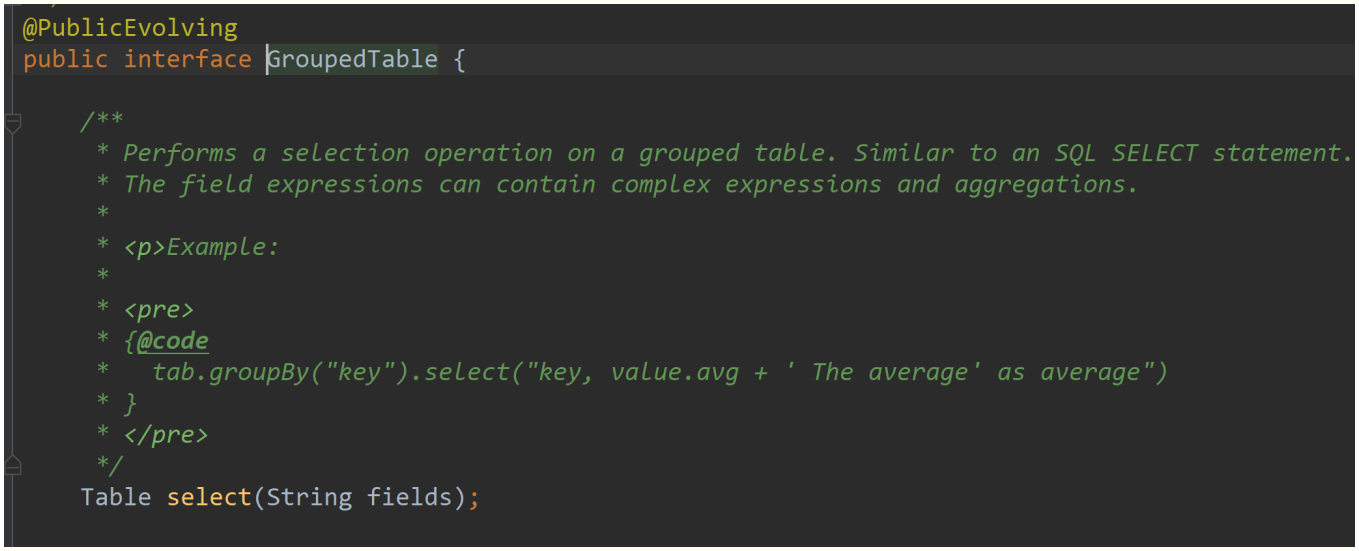
二、Flink Table & SQL 算子和内置函数

Table API 操作



以 Table -> GroupedTable -> Table 为例：





一个Table 经过 groupBy 后得到 GroupedTable，GroupedTable经过select后又得到Table。

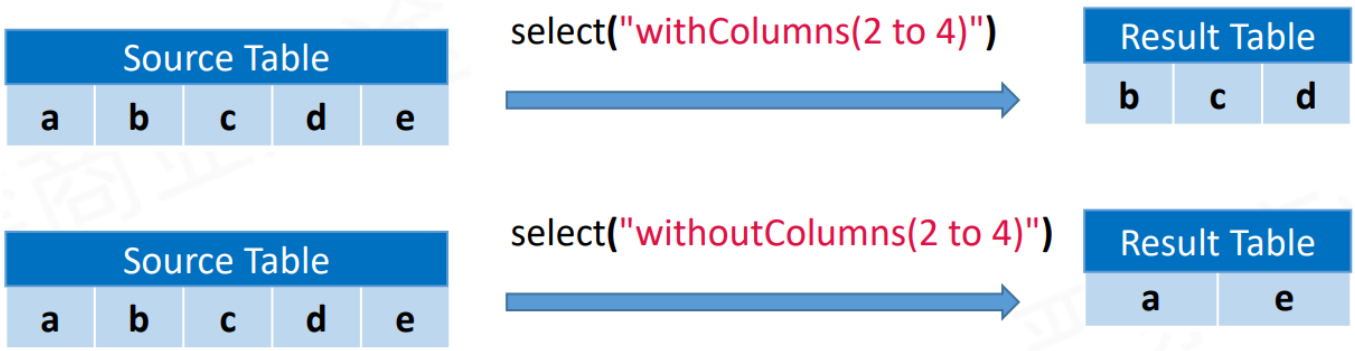
Columns operators

Operators	Examples
AddColumns	Table orders = tableEnv.scan("Orders"); Table result = orders.addColumns(“concat(c, ‘sunny‘) as desc");
AddOrReplaceColumns	Table orders = tableEnv.scan("Orders"); Table result = orders.addOrReplaceColumns("concat(c, 'sunny') as desc");
DropColumns	Table orders = tableEnv.scan("Orders"); Table result = orders.dropColumns("b, c");
RenameColumns	Table orders = tableEnv.scan("Orders"); Table result = orders.renameColumns("b as b2, c as c2");

比如有一张100列的表去除一列，可以选择 DropColumns

Columns Function

SYNTAX	DESC
withColumns(...)	select the specified columns
withoutColumns(...)	deselect the columns specified



Flink sql API操作

Flink SQL 和传统的 SQL 一样，支持了包含查询、连接、聚合等场景，另外还支持了包括窗口、排序等场景：

```
query:
  values
  | {
      select
      | selectWithoutFrom
      | query UNION [ ALL ] query
      | query EXCEPT query
      | query INTERSECT query
    }
    [ ORDER BY orderItem [, orderItem ]* ]
    [ LIMIT { count | ALL } ]
    [ OFFSET start { ROW | ROWS } ]
    [ FETCH { FIRST | NEXT } [ count ] { ROW | ROWS } ONLY]

orderItem:
  expression [ ASC | DESC ]

select:
  SELECT [ ALL | DISTINCT ]
  { * | projectItem [, projectItem ]* }
  FROM tableExpression
  [ WHERE booleanExpression ]
  [ GROUP BY { groupItem [, groupItem ]* } ]
  [ HAVING booleanExpression ]
  [ WINDOW windowName AS windowSpec [, windowName AS windowSpec ]* ]

selectWithoutFrom:
  SELECT [ ALL | DISTINCT ]
  { * | projectItem [, projectItem ]* }

projectItem:
  expression [ [ AS ] columnAlias ]
```

```
  | tableAlias . *

tableExpression:
  tableReference [, tableReference ]*
  | tableExpression [ NATURAL ] [ LEFT | RIGHT | FULL ] JOIN tableExpression [ joinCondition ]

joinCondition:
  ON booleanExpression
  | USING '(' column [, column ]* ')'

tableReference:
  tablePrimary
  [ matchRecognize ]
  [ [ AS ] alias [ '(' columnAlias [, columnAlias ]* ')' ] ]

tablePrimary:
  [ TABLE ] [ [ catalogName . ] schemaName . ] tableName
  | LATERAL TABLE '(' functionName '(' expression [, expression ]* ')' ')'
  | UNNEST '(' expression ')'

values:
  VALUES expression [, expression ]*

groupItem:
  expression
  | '(' ')'
  | '(' expression [, expression ]* ')'
  | CUBE '(' expression [, expression ]* ')'
  | ROLLUP '(' expression [, expression ]* ')'
  | GROUPING SETS '(' groupItem [, groupItem ]* ')'

windowRef:
    windowName
  | windowSpec

windowSpec:
    [ windowName ]
    '('
    [ ORDER BY orderItem [, orderItem ]* ]
    [ PARTITION BY expression [, expression ]* ]
    [
        RANGE numericOrIntervalExpression {PRECEDING}
      | ROWS numericExpression {PRECEDING}
    ]
    ')'
...
```

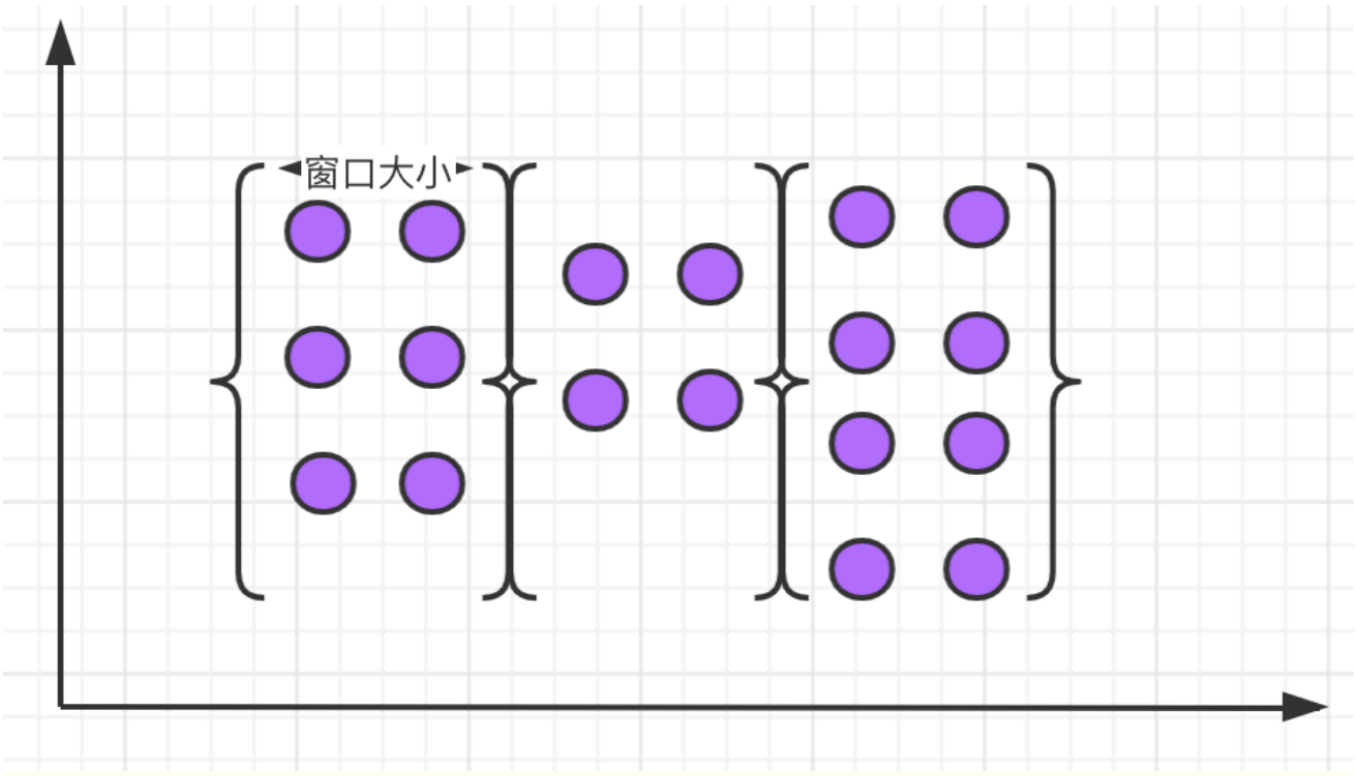
主要来看一下flink sql的Windows操作

根据窗口数据划分的不同，目前 Apache Flink 有如下 3 种：

- 滚动窗口，窗口数据有固定的大小，窗口中的数据不会叠加；
- 滑动窗口，窗口数据有固定大小，并且有生成间隔；
- 会话窗口，窗口数据没有固定的大小，根据用户传入的参数进行划分，窗口数据无叠加；

滚动窗口

- 滚动窗口的特点是：有固定大小、窗口中的数据不会重叠，如下图所示



语法：

```
SELECT
    [gk],
    [TUMBLE_START(timeCol, size)],
    [TUMBLE_END(timeCol, size)],
    agg1(col1),
    ...
    aggn(colN)
FROM Tab1
GROUP BY [gk], TUMBLE(timeCol, size)
```

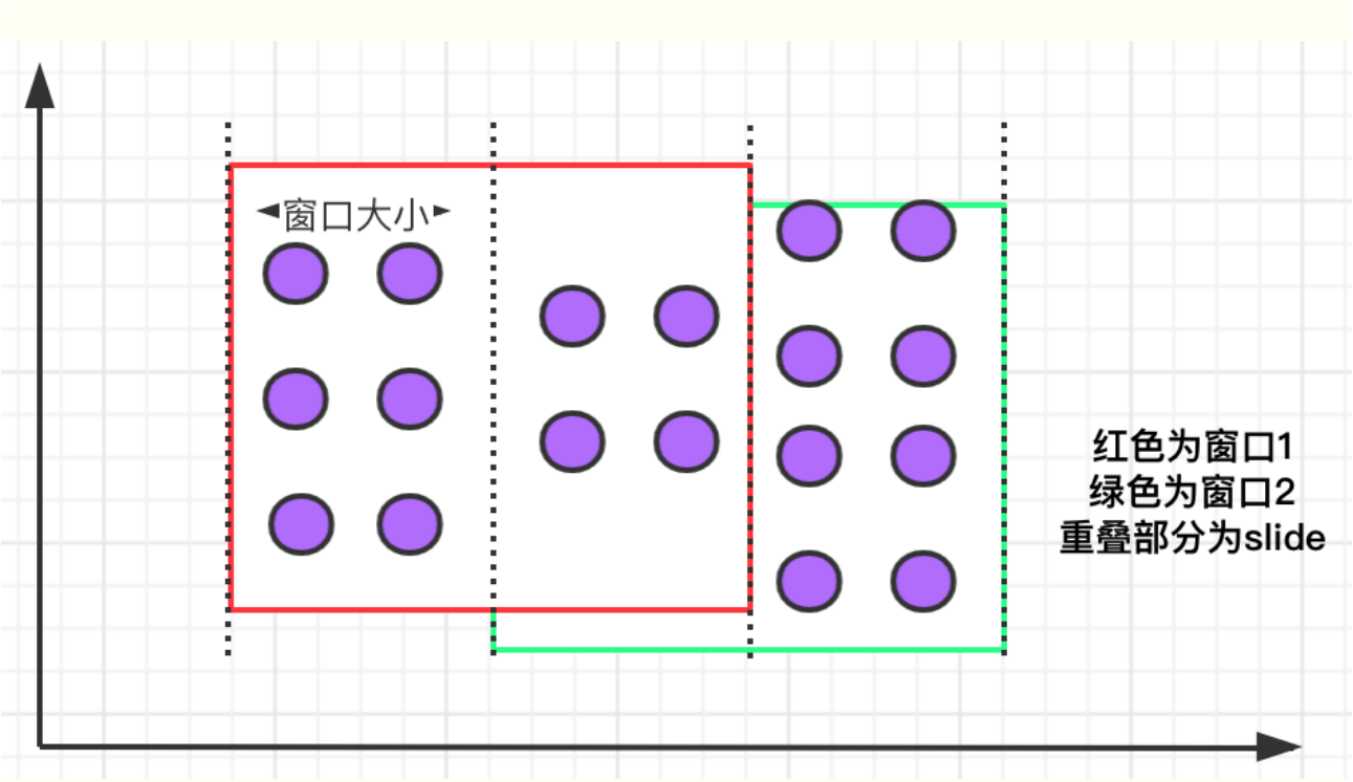
比如我们计算每个用户每天的订单量：

```
SELECT user, TUMBLE_START(timeLine, INTERVAL '1' DAY) as winStart, SUM(amount) FROM Orders GROU
P BY TUMBLE(timeLine, INTERVAL '1' DAY), user;
```

其中，TUMBLE_START 和 TUMBLE_END 代表窗口的开始时间和窗口的结束时间，TUMBLE (timeLine, INTERVAL '1' DAY) 中的 timeLine 代表时间字段所在的列，INTERVAL '1' DAY 表示时间间隔为一天。

滑动窗口

- 滑动窗口有固定的大小，与滚动窗口不同的是滑动窗口可以通过 slide 参数控制滑动窗口的创建频率。需要注意的是，多个滑动窗口可能会发生数据重叠，具体语义如下：



滑动窗口的语法与滚动窗口相比，只多了一个 slide 参数：

```
SELECT
    [gk],
    [HOP_START(timeCol, slide, size)] ,
    [HOP_END(timeCol, slide, size)],
    agg1(col1),
    ...
    aggN(colN)
FROM Tab1
GROUP BY [gk], HOP(timeCol, slide, size)
```

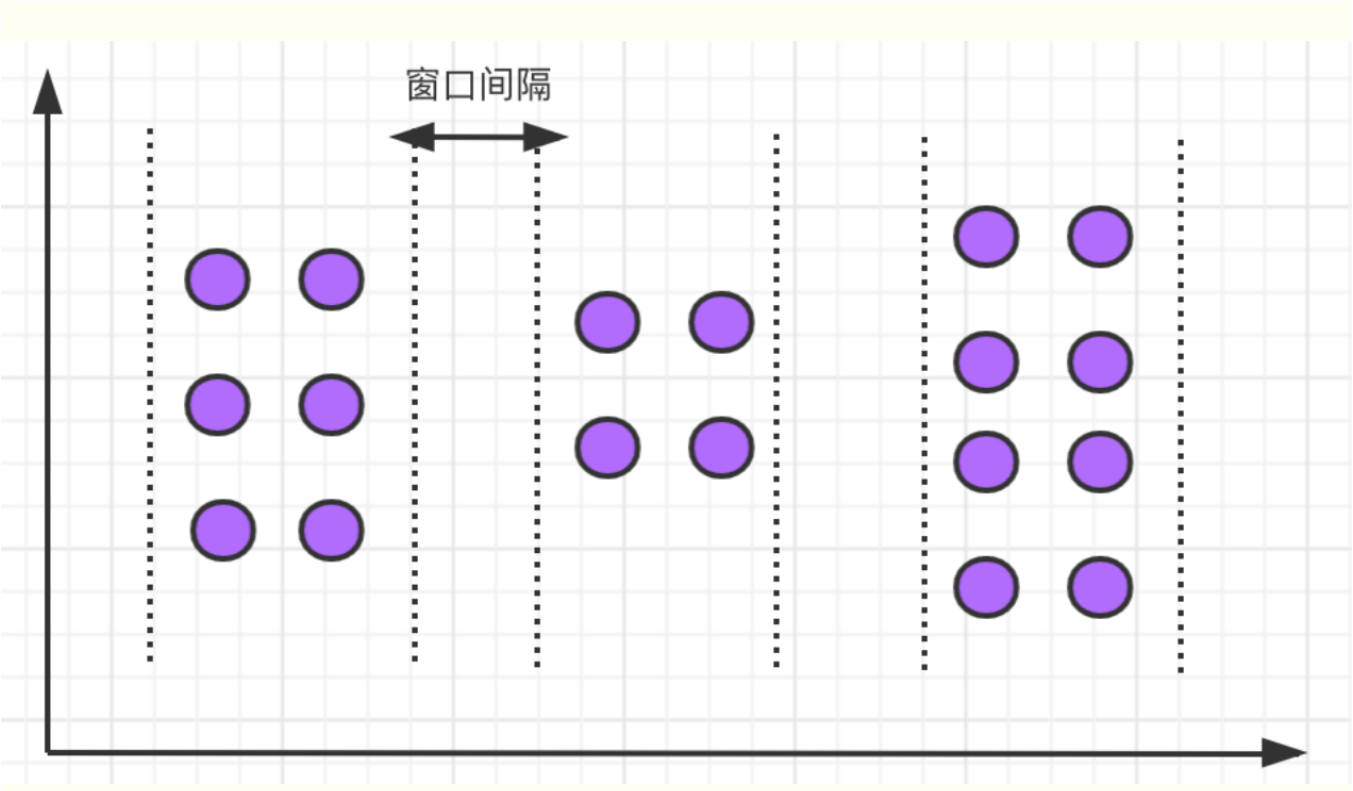
例如，我们要每间隔一小时计算一次过去 24 小时内每个商品的销量：

```
SELECT product, SUM(amount) FROM Orders GROUP BY HOP(rowtime, INTERVAL '1' HOUR, INTERVAL '1' DAY), product
```

上述案例中的 INTERVAL '1' HOUR 代表滑动窗口生成的时间间隔。

会话窗口

- 会话窗口定义了一个非活动时间，假如在指定的时间间隔内没有出现事件或消息，则会话窗口关闭。



语法如下：

```
SELECT
    [gk],
    SESSION_START(timeCol, gap) AS winStart,
    SESSION_END(timeCol, gap) AS winEnd,
    agg1(col1),
     ...
    aggn(colN)
FROM Tab1
GROUP BY [gk], SESSION(timeCol, gap)
```

举例，我们需要计算每个用户过去 1 小时内的订单量：

```
SELECT user, SESSION_START(rowtime, INTERVAL '1' HOUR) AS sStart, SESSION_ROWTIME(rowtime, INTERVAL '1' HOUR) AS sEnd, SUM(amount) FROM Orders GROUP BY SESSION(rowtime, INTERVAL '1' HOUR), user
```

内置函数

Flink 中还有大量的内置函数，我们可以直接使用，将内置函数分类如下：

- 比较函数

函数	逻辑描述
value1=value2	如果 value1 等于 value2，则返回 TRUE ; 如果 value1 或 value2 为 NULL，则返回 UNKNOWN
value1<>value2	如果 value1 不等于 value2，则返回 TRUE ; 如果 value1 或 value2 为 NULL，则返回 UNKNOWN
value1>value2	如果 value1 大于 value2，则返回 TRUE ; 如果 value1 或 value2 为 NULL，则返回 UNKNOWN
value1 <value2	如果 value1 小于 value2，则返回 TRUE ; 如果 value1 或 value2 为 NULL，则返回 UNKNOWN
value IS NULL	如果 value 为 NULL，则返回 TRUE
value IS NOT NULL	如果 value 不为 NULL，则返回 TRUE
string1 LIKE string2	如果 string1 匹配模式 string2，则返回 TRUE ; 如果 string1 或 string2 为 NULL，则返回 UNKNOWN
value1 IN (value2, value3…)	如果给定列表中存在 value1 （value2，value3，…），则返回 TRUE 。当（value2，value3，…）包含 NULL，如果可以找到该数据元则返回 TRUE，否则返回 UNKNOWN；如果 value1 为 NULL，则始终返回 UNKNOWN

- 逻辑函数

函数	逻辑描述
A OR B	如果 A 为 TRUE 或 B 为 TRUE，则返回 TRUE
A AND B	如果 A 和 B 都为 TRUE，则返回 TRUE
NOT boolean	如果 boolean 为 FALSE，则返回 TRUE，否则返回 TRUE如果 boolean 为 TRUE，则返回 FALSE
A IS TRUE 或 FALSE	判断 A 是否为真

- 算数函数

函数	逻辑描述
numeric1 ±*/ numeric2	分别代表两个数值加减乘除
ABS(numeric)	返回 numeric 的绝对值
POWER(numeric1, numeric2)	返回 numeric1 上升到 numeric2 的幂

- 字符串处理函数

函数	逻辑描述
UPPER/LOWER	以大写 / 小写形式返回字符串
LTRIM(string)	返回一个字符串，从去除左空格的字符串 类似还有 RTRIM
CONCAT(string1, string2,…)	返回连接 string1、string2、… 的字符串

- 时间函数

函数	逻辑描述
DATE string	返回以“yyyy-MM-dd”形式从字符串解析的 SQL 日期
TIMESTAMP string	返回以字符串形式解析的 SQL 时间戳，格式为“yyyy-MM-dd HH：mm：ss [.SSS]”
CURRENT_DATE	返回 UTC 时区中的当前 SQL 日期
DATE_FORMAT(timestamp, string)	返回使用指定格式字符串格式化时间戳的字符串

三、demo

代码地址

```
import org.apache.flink.api.scala._
import org.apache.flink.configuration.{ConfigConstants, Configuration}
import org.apache.flink.streaming.api.functions.source.SourceFunction
import org.apache.flink.streaming.api.scala.{DataStream, SplitStream, StreamExecutionEnvironment}
import org.apache.flink.table.api.scala.StreamTableEnvironment
import org.apache.flink.table.api.{StreamQueryConfig, Table}

import scala.collection.mutable.{ArrayBuffer, ListBuffer}
import scala.util.Random

/**
  * @author xiandongxie
  */

// 商品类
case class Item(id: Int, name: String)

// 自定义实时数据源
class MyStreamSourceFunction extends SourceFunction[Item] {

  var isCancel: Boolean = false

  override def cancel(): Unit = {
    isCancel = true
  }
```

```
  override def run(ctx: SourceFunction.SourceContext[Item]): Unit = {
    while (!isCancel) {
      val item: Item = getItem()
      ctx.collect(item)
      Thread.sleep(1000)
    }
  }

  def getItem(): Item = {
    val random: Random = new Random()
    val id: Int = random.nextInt(100)
    val buffer: ArrayBuffer[String] = new ArrayBuffer[String]()
    buffer.append("HAT")
    buffer.append("TIE")
    buffer.append("SHOE")
    new Item(id, buffer(random.nextInt(3)))

  }

}


/**
  * 我们把实时的商品数据流进行分流，分成 even 和 odd 两个流进行 JOIN，条件是名称相同
  * 最后，把两个流的 JOIN 结果输出
  */
object StreamJoinDemo {
  def main(args: Array[String]): Unit = {
    val logPath: String = "/tmp/logs/flink_log"

    // 生成配置对象
    var conf: Configuration = new Configuration()
    // 开启flink web UI
    conf.setBoolean(ConfigConstants.LOCAL_START_WEBSERVER, true)
    // 配置web UI的日志文件，否则打印日志到控制台
    conf.setString("web.log.path", logPath)
    // 配置taskManager的日志文件，否则打印到控制台
    conf.setString(ConfigConstants.TASK_MANAGER_LOG_PATH_KEY, logPath)
    // 获取local运行环境
    val streamEnv: StreamExecutionEnvironment = StreamExecutionEnvironment.createLocalEnvironme
ntWithWebUI(conf)
    //    val build: EnvironmentSettings = EnvironmentSettings.newInstance.useBlinkPlanner.inSt
reamingMode.build
    //    val tableEnv: StreamTableEnvironment = StreamTableEnvironment.create(streamEnv, build
)
    val tableEnv: StreamTableEnvironment = StreamTableEnvironment.create(streamEnv)

    val source: DataStream[Item] = streamEnv.addSource(new MyStreamSourceFunction)
    val split: SplitStream[Item] = source.split(f => {
      val out: ListBuffer[String] = new ListBuffer[String]
      val id: Int = f.id
      if (id % 2 == 0) {
        out.append("even")
      } else {
        out.append("odd")
```

```
      }
      out
    })

    val evenData: DataStream[Item] = split.select("even")
    val oddData: DataStream[Item] = split.select("odd")

    tableEnv.createTemporaryView("event_table", evenData)
    tableEnv.createTemporaryView("odd_table", oddData)

    val queryTable: Table = tableEnv.sqlQuery("select a.id,a.name,b.id,b.name from event_table
as a join odd_table as b on a.name = b.name")
    queryTable.printSchema()
    tableEnv.toRetractStream[(Int, String, Int, String)](queryTable, new StreamQueryConfig())
      .print()


    streamEnv.execute()
  }
}
```

结果:

```
root
 |-- id: INT
 |-- name: STRING
 |-- id0: INT
 |-- name0: STRING

20/04/28 22:08:23 WARN WebMonitorUtils: Log file en
6> (true,(4,HAT,97,HAT))
8> (true,(22,TIE,59,TIE))
6> (true,(68,HAT,97,HAT))
20/04/28 22:08:32 ERROR JobDetailsHandler: Exceptio
 not found
6> (true,(4,HAT,3,HAT))
6> (true,(68,HAT,3,HAT))
8> (true,(74,SHOE,55,SHOE))
8> (true,(74,SHOE,67,SHOE))
6> (true,(90,HAT,3,HAT))
6> (true,(90,HAT,97,HAT))
8> (true,(74,SHOE,53,SHOE))
8> (true,(88,SHOE,53,SHOE))
```