

# Hudi

## 第一章 数据湖的介绍

---

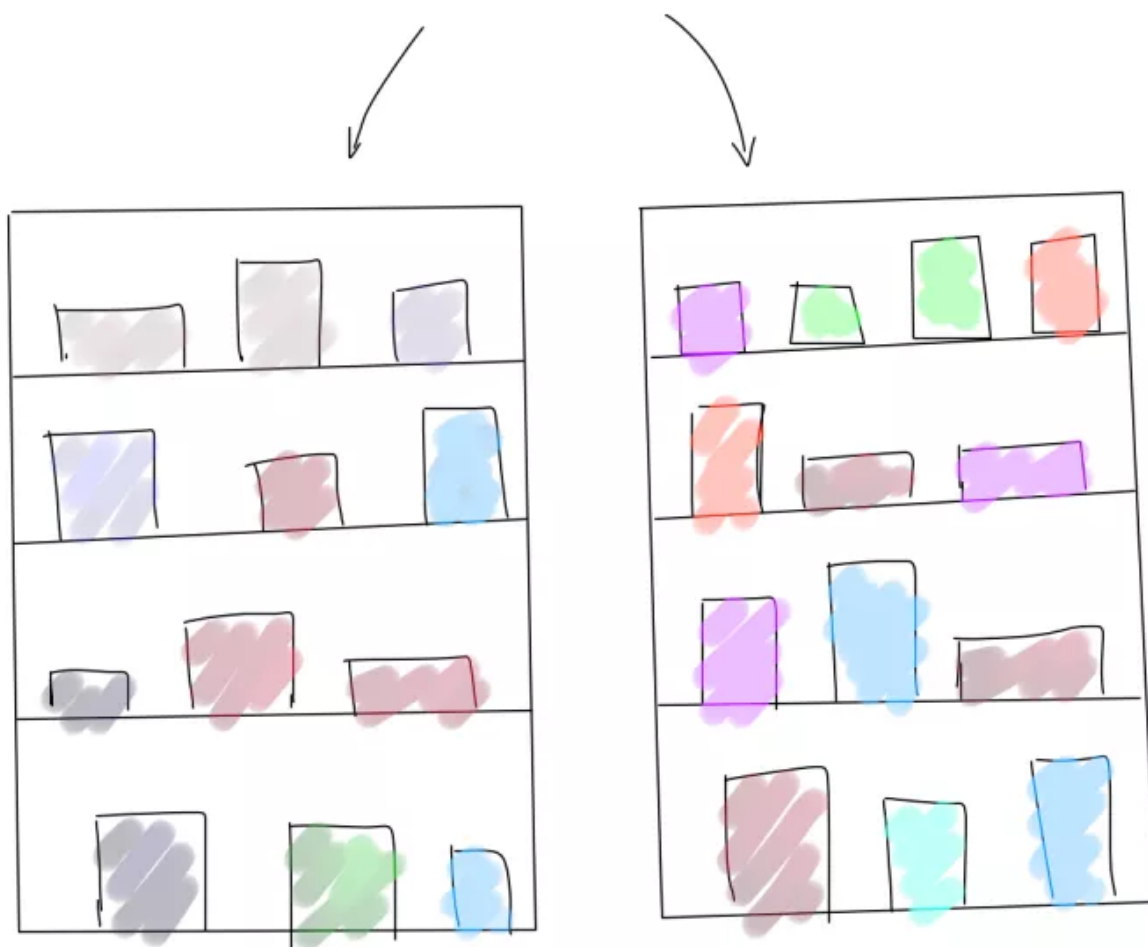
### 数据湖Data Lake

数据湖是大数据架构的新范式，以原始格式存储数据，可以满足用户的广泛需求，并能提供更快洞察力，细致的数据编录和管理是成功实施数据湖的关键。

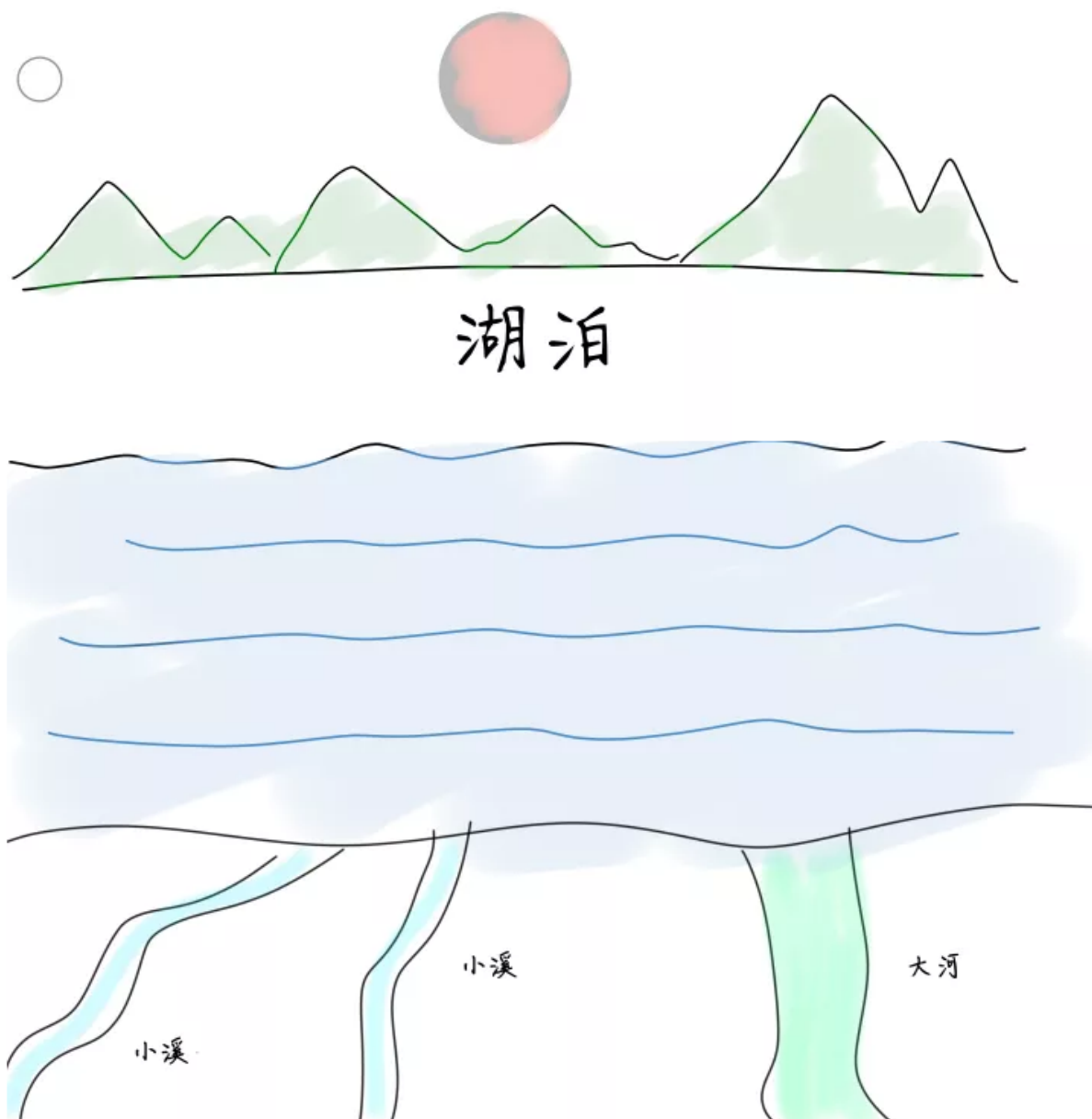
### 仓库和湖泊

仓库（WareHouse）是人为提前建造好的，有货架，还有过道，并且还可以进一步为放置到货架的物品指定位置。

货架上可以包含很多的货物  
都是有结构的方式放置的



而湖泊（Lake）是液态的，是不断变化的、没有固定形态的，基本上是没有结构的，湖泊可以由河流、小溪和其他未被任何处理的水源维持。湖泊是不需要预先指定结构的。

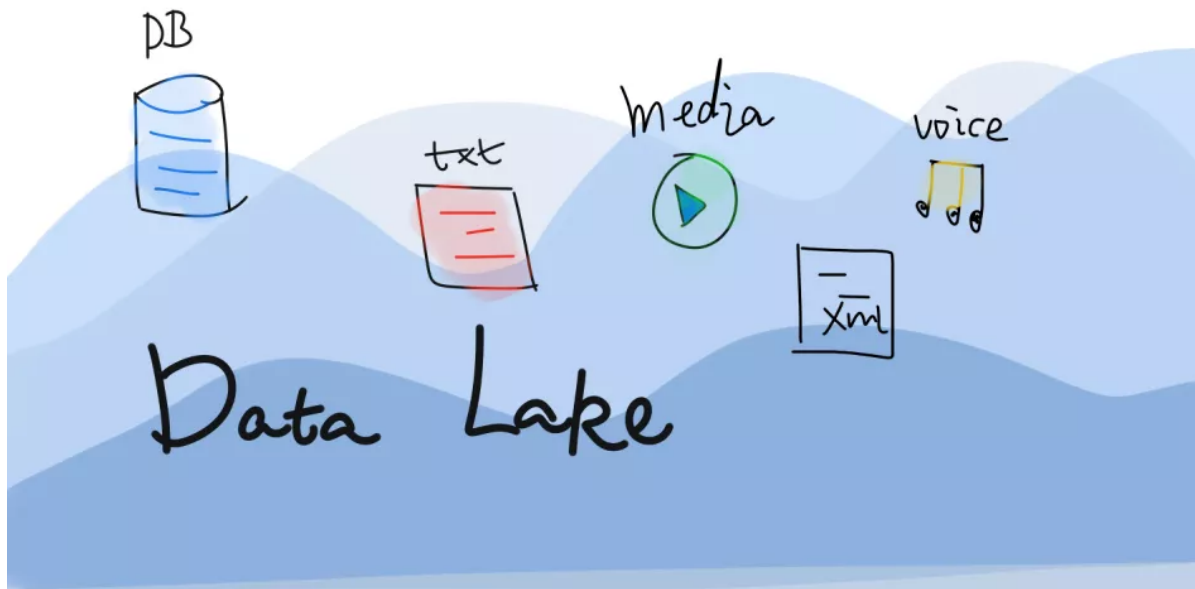


## 什么是数据湖

Data lake这个术语由Pentaho公司的创始人兼首席技术官詹姆斯·狄克逊(James Dixon)提出，他对数据湖的解释是：把你以前在磁带上拥有的东西倒入到数据湖，然后开始探索该数据。

数据湖（Data Lake）和数据库、数据仓库一样，都是数据存储的设计模式。数据库和数据仓库会以关系型的方式来设计存储、处理数据。但数据湖的设计理念是相反的，数据仓库是为了保障数据的质量、数据的一致性、数据的重用性等对数据进行结构化处理。

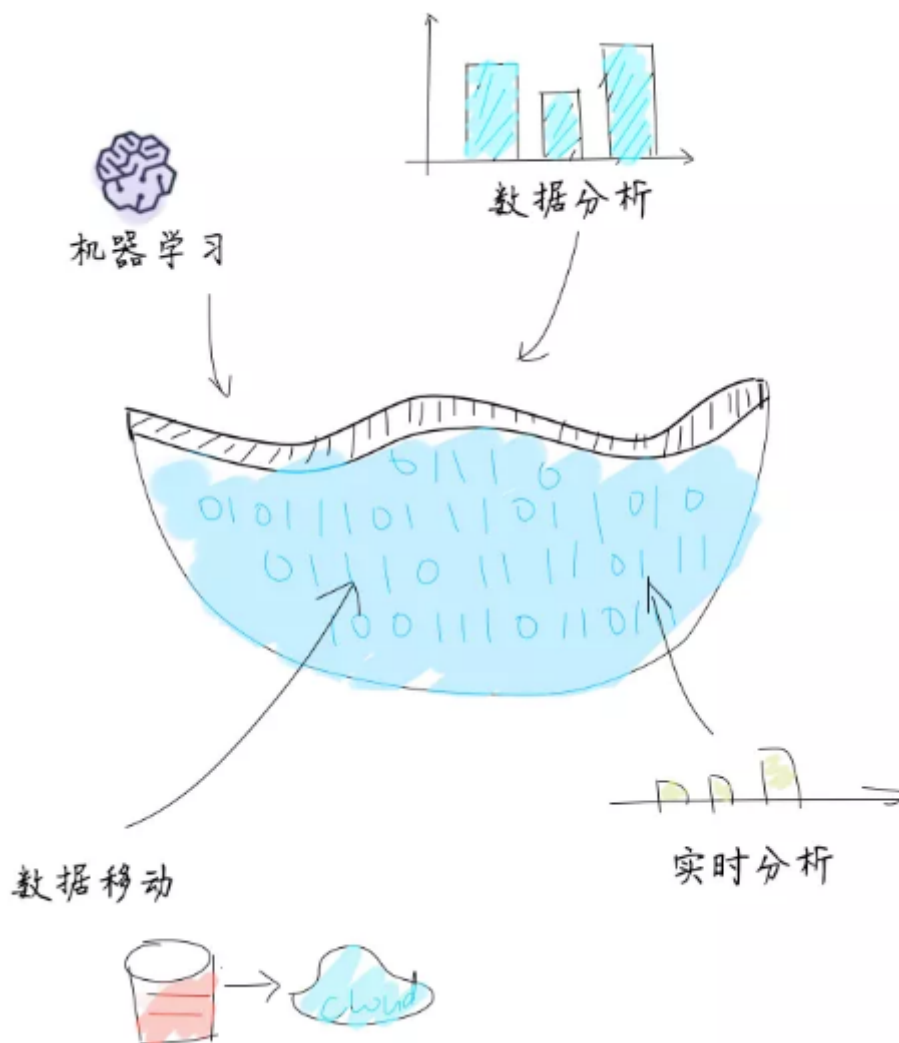
数据湖是一个数据存储库，可以使用数据湖来存储大量的原始数据。现在企业的数据仓库都会通过分层的方式将数据存储于文件夹、文件中，而数据湖使用的是平面架构来存储数据。我们需要做的只是给每个数据元素分配一个唯一的标识符，并通过元数据标签来进行标注。当企业中出现业务问题时，可以从数据湖中查询数据，然后分析业务对应的那一小部分数据集来解决业务问题。



了解过Hadoop的同学知道，基于Hadoop可以存储任意形式的数据。所以，很多时候数据湖会和Hadoop关联到一起。例如：把数据加载Hadoop中，然后将数据分析、和数据挖掘的工具基于Hadoop进行处理。

数据湖越来越多的用于描述任何的大型数据池，数据都是以原始数据方式存储，知道需要查询应用数据的时候才会开始分析数据需求和应用架构。

数据湖是专注于原始数据保存以及低成本长期存储的存储设计模式，它相当于是对数据仓库的补充。数据湖是用于长期存储数据容器的集合，通过数据湖可以大规模的捕获、加工、探索任何形式的原始数据。通过使用一些低成本的技术，可以让下游设施可以更好地利用，下游设施包括像数据集市、数据仓库或者是机器学习模型。



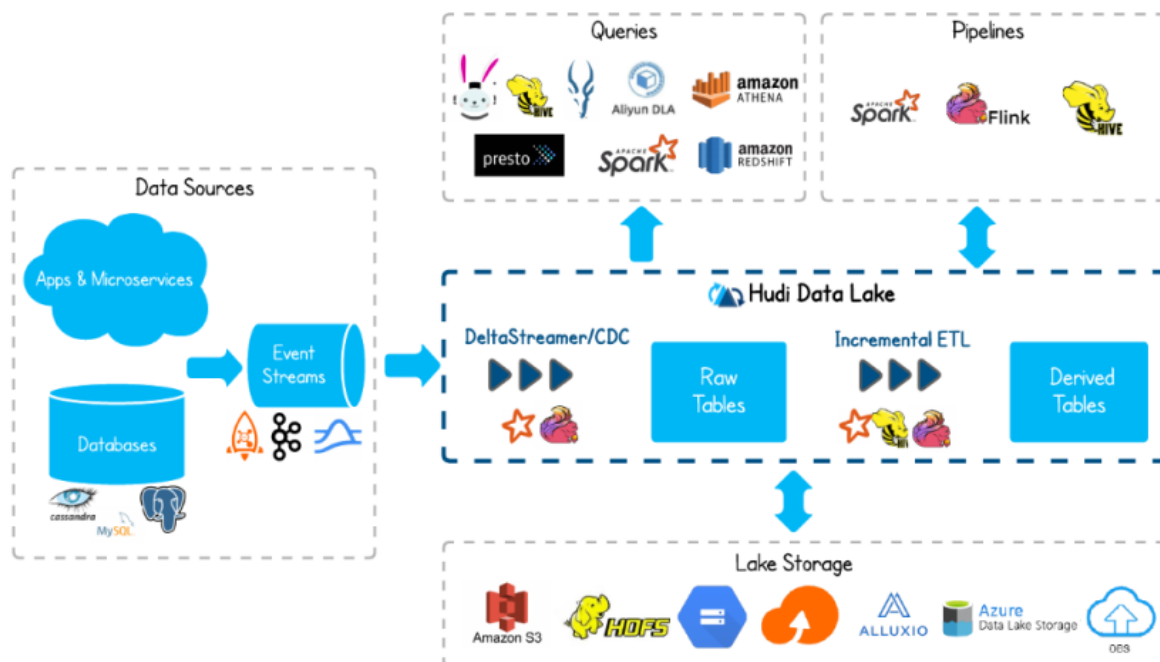
## 第二章 Hudi概述

### 简介

Apache Hudi (Hadoop Upserts Delete and Incremental) 是下一代 **流数据湖平台**。Apache Hudi将核心仓库和数据库功能直接引入数据湖。Hudi提供了表、事务、高效的upserts/delete、高级索引、流摄取服务、数据集群/压缩优化和并发，同时保持数据的开源文件格式。

Apache Hudi不仅非常适合于流工作负载，而且还允许创建高效的增量批处理管道。

Apache Hudi可以轻松地在任何云存储平台上使用。Hudi的高级性能优化，使分析工作负载更快的任何流行的查询引擎，包括Apache Spark、Flink、Presto、Trino、Hive等。



## 发展历史

- 2015 年：发表了增量处理的核心思想/原则（O'reilly 文章）。
- 2016 年：由 Uber 创建并为所有数据库/关键业务提供支持。
- 2017 年：由 Uber 开源，并支撑 100PB 数据湖。
- 2018 年：吸引大量使用者，并因云计算普及。
- 2019 年：成为 ASF 孵化项目，并增加更多平台组件。
- 2020 年：毕业成为 Apache 顶级项目，社区、下载量、采用率增长超过 10 倍。
- 2021 年：支持 Uber 500PB 数据湖，SQL DML、Flink 集成、索引、元服务器、缓存。

## Hudi特性

可插拔索引机制支持快速Upsert/Delete。

支持增量拉取表变更以进行处理。

支持事务提交及回滚，并发控制。

支持Spark、Presto、Trino、Hive、Flink等引擎的SQL读写。

自动管理小文件，数据聚簇，压缩，清理。

流式摄入，内置CDC源和工具。

内置可扩展存储访问的元数据跟踪。

向后兼容的方式实现表结构变更的支持。

## 使用场景

### 1) 近实时写入

减少碎片化工具的使用。

CDC 增量导入 RDBMS 数据。

限制小文件的大小和数量。

### 2) 近实时分析

相对于秒级存储（Druid, OpenTSDB），节省资源。

提供分钟级别时效性，支撑更高效的查询。

Hudi作为lib，非常轻量。

### 3) 增量 pipeline

区分arrivetime和event time处理延迟数据。

更短的调度interval减少端到端延迟（小时 -> 分钟） => Incremental Processing。

### 4) 增量导出

替代部分Kafka的场景，数据导出到在线服务存储。

## 第三章 利用IDEA开发Hudi

Apache Hudi最初是由Uber开发的，旨在以高效率实现低延迟的数据库访问。Hudi 提供了Hudi 表的概念，这些表支持CRUD操作。接下来，[基于Spark框架使用Hudi API 进行读写操作](#)。

```
1 package com.lwPigKing.hudi.spark
2
3 import org.apache.hudi.QuickstartUtils._
4 import org.apache.spark.sql.functions.col
5 import org.apache.spark.sql.{DataFrame, Dataset, Row, SaveMode, SparkSession}
```

```

6
7 import java.util
8
9
10 /**
11  * Project:  BigDataProject
12  * Create date:  2023/8/7
13  * Created by lwPigKing
14  */
15 object HudiSparkDemo {
16     def main(args: Array[String]): Unit = {
17
18         val sparkSession: SparkSession = SparkSession
19             .builder()
20             .appName(this.getClass.getSimpleName.stripSuffix("$"))
21             .master("local[*]")
22             .config("spark.serializer", "org.apache.spark.serializer.KryoSerializer")
23             .getOrCreate()
24
25
26         val tableName: String = "tbl_trips_cow"
27         val tablePath: String = "/hudi-warehouse/tbl_trips_cow"
28
29         // build data generators that simulate inserting and updating data
30         import org.apache.hudi.QuickstartUtils._
31
32
33         // Task1:Simulate data,insert hudi table and use COW mode
34         insertData(sparkSession, tableName, tablePath)
35
36         // Task2:Snapshot Query data in DSL mode
37         queryData(sparkSession, tablePath)
38         queryDataTime(sparkSession, tablePath)
39
40         // // Task3:Update the data
41         val generator: DataGenerator = new DataGenerator()
42         insertData(sparkSession, tableName, tablePath, generator)
43         updateData(sparkSession, tableName, tablePath, generator)
44         //
45         // // Task4:Incremental Query data in SQL
46         incrementalQueryData(sparkSession, tablePath)
47         //
48         // // Task5:Delete the data
49         deleteData(sparkSession, tableName, tablePath)
50
51
52         sparkSession.close()
53
54
55     }
56 }
57
58
59 def insertData(sparkSession: SparkSession, table: String, path: String): Unit = {
60     import sparkSession.implicitly._
61     import org.apache.hudi.QuickstartUtils._
62
63     val generator: DataGenerator = new DataGenerator

```

```

64     val inserts: util.List[String] = convertToStringList(generator.generateInserts(100))
65
66     import scala.collection.JavaConverters._
67     val insertDF: DataFrame = sparkSession
68         .read
69         .json(sparkSession.sparkContext.parallelize(inserts.asScala, 2).toDS())
70
71     import org.apache.hudi.DataSourceWriteOptions._
72     import org.apache.hudi.config.HoodieWriteConfig._
73     insertDF.write
74         .mode(SaveMode.Append)
75         .format("hudi")
76         .option("hoodie.insert.shuffle.parallelism", "2")
77         .option("hoodie.upsert.shuffle.parallelism", "2")
78         .option(PRECOMBINE_FIELD.key(), "ts")
79         .option(RECORDKEY_FIELD.key(), "uuid")
80         .option(PARTITIONPATH_FIELD.key(), "partitionpath")
81         .option(TBL_NAME.key(), table)
82         .save(path)
83
84 }
85
86
87 def queryData(sparkSession: SparkSession, path: String): Unit = {
88     import sparkSession.implicits._
89     val tripsDF: DataFrame = sparkSession.read.format("hudi").load(path)
90     tripsDF.filter(col("fare") >= 20 && col("fare") <= 50)
91         .select($"driver", $"rider", $"fare", $"begin_lat", $"begin_log", $"partitionpath",
92             $"_hoodie_commit_time")
93         .orderBy($"fare".desc, $"_hoodie_commit_time".desc)
94         .show(20, truncate = false)
95 }
96
97 def queryDataTime(sparkSession: SparkSession, path: String): Unit = {
98     import org.apache.spark.sql.functions._
99
100     // method 1: specify a string in the format yyyyMMddHHmmss
101     val df1: Dataset[Row] = sparkSession.read
102         .format("hudi")
103         .option("as.of.instant", "20211119095057")
104         .load(path)
105         .sort(col("_hoodie_commit_time").desc)
106     df1.show(numRows = 5, truncate = false)
107
108     // method 2: specify a string in the format yyyy-MM-dd HH:mm:ss
109     val df2: Dataset[Row] = sparkSession.read
110         .format("hudi")
111         .option("as.of.instant", "20211119095057")
112         .load(path)
113         .sort(col("_hoodie_commit_time").desc)
114     df2.show(numRows = 5, truncate = false)
115 }
116
117 def insertData(sparkSession: SparkSession, table: String, path: String, dataGen: DataGenerator):
118     Unit = {
119     import sparkSession.implicits._

```



```

120 // TODO: a. 模拟乘车数据
121 import org.apache.hudi.QuickstartUtils._
122 val inserts = convertToStringList(dataGen.generateInserts(100))
123
124 import scala.collection.JavaConverters._
125 val insertDF: DataFrame = sparkSession.read
126   .json(sparkSession.sparkContext.parallelize(inserts.asScala, 2).toDS())
127 //insertDF.printSchema()
128 //insertDF.show(10, truncate = false)
129
130 // TODO: b. 插入数据至Hudi表
131 import org.apache.hudi.DataSourceWriteOptions._
132 import org.apache.hudi.config.HoodieWriteConfig._
133 insertDF.write
134   .mode(SaveMode.Overwrite)
135   .format("hudi") // 指定数据源为Hudi
136   .option("hoodie.insert.shuffle.parallelism", "2")
137   .option("hoodie.upsert.shuffle.parallelism", "2")
138   // Hudi 表的属性设置
139   .option(PRECOMBINE_FIELD.key(), "ts")
140   .option(RECORDKEY_FIELD.key(), "uuid")
141   .option(PARTITIONPATH_FIELD.key(), "partitionpath")
142   .option(TBL_NAME.key(), table)
143   .save(path)
144 }
145
146 def updateData(sparkSession: SparkSession, table: String, path: String, dataGen: DataGenerator):
Unit = {
147   import sparkSession.implicits._
148   import org.apache.hudi.QuickstartUtils._
149   import scala.collection.JavaConverters._
150   val updates: util.List[String] = convertToStringList(dataGen.generateUpdates(100))
151   val updateDF: DataFrame = sparkSession.read
152     .json(sparkSession.sparkContext.parallelize(updates.asScala, 2).toDS())
153
154   import org.apache.hudi.DataSourceWriteOptions._
155   import org.apache.hudi.config.HoodieWriteConfig._
156   updateDF.write
157     .mode(SaveMode.Append)
158     .format("hudi")
159     .option("hoodie.insert.shuffle.parallelism", "2")
160     .option("hoodie.upsert.shuffle.parallelism", "2")
161     .option(PRECOMBINE_FIELD.key(), "ts")
162     .option(RECORDKEY_FIELD.key(), "uuid")
163     .option(PARTITIONPATH_FIELD.key(), "partitionpath")
164     .option(TBL_NAME.key(), table)
165     .save(path)
166 }
167
168
169 def incrementalQueryData(sparkSession: SparkSession, path: String): Unit = {
170   import sparkSession.implicits._
171   import org.apache.hudi.DataSourceReadOptions._
172   sparkSession.read
173     .format("hudi")
174     .load(path)
175     .createOrReplaceTempView("view_temp_hudi_trips")
176

```

```

177     val commits: Array[String] = sparkSession.sql(
178         s"""
179             |select
180             |   distinct(_hoodie_commit_time) as commitTime
181             |from
182             |   view_temp_hudi_trips
183             |order by
184             |   commitTime DESC
185             |""".stripMargin)
186     .map(row => {
187         row.getString(0)
188     }).take(50)
189
190     val beginTime: String = commits(commits.length - 1)
191     println(s"beginTime = ${beginTime}")
192
193     val tripsIncrementalDF: DataFrame = sparkSession.read
194         .format("hudi")
195         .option(QUERY_TYPE.key(), QUERY_TYPE_INCREMENTAL_OPT_VAL)
196         .option(BEGIN_INSTANTTIME.key(), beginTime)
197         .load(path)
198
199     tripsIncrementalDF.createOrReplaceTempView("hudi_trips_incremental")
200     sparkSession.sql(
201         s"""
202             |select
203             |   _hoodie_commit_time, fare, begin_lon, begin_lat, ts
204             |from
205             |   hudi_trips_incremental
206             |where
207             |   fare > 20.0
208             |""".stripMargin)
209         .show(10, truncate = false)
210     }
211
212
213     def deleteData(sparkSession: SparkSession, table: String, path: String): Unit = {
214         import sparkSession.implicit._
215         val tripsDF: DataFrame = sparkSession.read.format("hudi").load(path)
216         println(s"Count = ${tripsDF.count()}")
217
218         val value: Dataset[Row] = tripsDF.select($"uuid", $"partitionpath").limit(2)
219         import org.apache.hudi.QuickstartUtils._
220
221         val generator: DataGenerator = new DataGenerator()
222         val deletes: util.List[String] = generator.generateDeletes(value.collectAsList())
223
224         import scala.collection.JavaConverters._
225         val deleteDF: DataFrame =
226             sparkSession.read.json(sparkSession.sparkContext.parallelize(deletes.asScala, 2))
227
228         import org.apache.hudi.DataSourceWriteOptions._
229         import org.apache.hudi.config.HoodieWriteConfig._
230         deleteDF.write
231             .mode(SaveMode.Append)
232             .format("hudi")
233             .option("hoodie.insert.shuffle.parallelism", "2")
234             .option("hoodie.upsert.shuffle.parallelism", "2")

```

```

234     .option(OPERATION.key(), "delete")
235     .option(PRECOMBINE_FIELD.key(), "ts")
236     .option(RECORDKEY_FIELD.key(), "uuid")
237     .option(PARTITIONPATH_FIELD.key(), "partitionpath")
238     .option(TBL_NAME.key(), table)
239     .save(path)
240
241     val hudiDF: DataFrame = sparkSession.read.format("hudi").load(path)
242     println(s"Delete after count = ${hudiDF.count()}")
243 }
244

```

## 第四章 FlinkSQL开发Hudi

### 1 读数据

```

1     package com.lwPigKing.hudi.flink;
2
3     import org.apache.flink.table.api.EnvironmentSettings;
4     import org.apache.flink.table.api.TableEnvironment;
5
6     /**
7      * Project:   BigDataProject
8      * Create date: 2023/8/8
9      * Created by lwPigKing
10     */
11     public class FlinkSQLReadDemo {
12         public static void main(String[] args) {
13             EnvironmentSettings settings =
14             EnvironmentSettings.newInstance().inStreamingMode().build();
15             TableEnvironment tableEnv = TableEnvironment.create(settings);
16
17             tableEnv.executeSql(
18                 "CREATE TABLE order_hudi(\n" +
19                     "    orderId STRING PRIMARY KEY NOT ENFORCED,\n" +
20                     "    userId STRING,\n" +
21                     "    orderTime STRING,\n" +
22                     "    ip STRING,\n" +
23                     "    orderMoney DOUBLE,\n" +
24                     "    orderStatus INT,\n" +
25                     "    ts STRING,\n" +
26                     "    partition_day STRING\n" +
27                     ")\n" +
28                     "PARTITIONED BY (partition_day)\n" +
29                     "WITH (\n" +
30                     "    'connector' = 'hudi',\n" +
31                     "    'path' = 'file:///D:/flink_hudi_order',\n" +
32                     "    'table.type' = 'MERGE_ON_READ',\n" +
33                     "    'read.streaming.enabled' = 'true',\n" +
34                     "    'read.streaming.check-interval' = '4'\n" +
35                     ")\n" +
36             );
37

```

```

37         tableEnv.executeSql(
38             "SELECT\n" +
39                 "orderId, userId, orderTime, ip, orderMoney, orderStatus, ts,\n" +
40                 "partition_day\n" +
41                 "FROM order_hudi"
42         ).print();
43     }
44 }
45

```

## 2 插数据

```

1  package com.lwPigKing.hudi.flink;
2
3  /**
4   * Project:   BigDataProject
5   * Create date: 2023/8/8
6   * Created by lwPigKing
7   */
8
9  import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
10 import org.apache.flink.table.api.EnvironmentSettings;
11 import org.apache.flink.table.api.Table;
12 import org.apache.flink.table.api.bridge.java.StreamTableEnvironment;
13
14 import static org.apache.flink.table.api.Expressions.$;
15
16 /**
17  * Based on Flink SQL: the data in the topic is consumed in real time,
18  * and after conversion processing, it's stored in the Hudi table in real time
19  */
20 public class FlinkSQLHudiDemo {
21     public static void main(String[] args) {
22         StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
23         env.setParallelism(1);
24         env.enableCheckpointing(5000);
25         EnvironmentSettings settings =
26             EnvironmentSettings.newInstance().inStreamingMode().build();
27         StreamTableEnvironment tableEnv = StreamTableEnvironment.create(env, settings);
28
29         tableEnv.executeSql(
30             "CREATE TABLE order_kafka_source (\n" +
31                 "    orderId STRING,\n" +
32                 "    userId STRING,\n" +
33                 "    orderTime STRING,\n" +
34                 "    ip STRING,\n" +
35                 "    orderMoney DOUBLE,\n" +
36                 "    orderStatus INT\n" +
37                 ") WITH (\n" +
38                 "    'connector' = 'kafka',\n" +
39                 "    'topic' = 'order-topic',\n" +
40                 "    'properties.bootstrap.servers' = 'node1.itcast.cn:9092',\n" +
41                 "    'properties.group.id' = 'gid-1001',\n" +

```

```

41         " 'scan.startup.mode' = 'latest-offset',\n" +
42         " 'format' = 'json',\n" +
43         " 'json.fail-on-missing-field' = 'false',\n" +
44         " 'json.ignore-parse-errors' = 'true'\n" +
45         ")\n"
46     );
47
48     Table etlTable = tableEnv
49         .from("order_kafka_source")
50         .addColumns(
51             $("orderId").substring(0, 17).as("ts")
52         )
53         .addColumns(
54             $("orderTime").substring(0, 10).as("partition_day")
55         );
56     tableEnv.createTemporaryView("view_order", etlTable);
57
58     tableEnv.executeSql(
59         "CREATE TABLE order_hudi_sink (\n" +
60         "   orderId STRING PRIMARY KEY NOT ENFORCED,\n" +
61         "   userId STRING,\n" +
62         "   orderTime STRING,\n" +
63         "   ip STRING,\n" +
64         "   orderMoney DOUBLE,\n" +
65         "   orderStatus INT,\n" +
66         "   ts STRING,\n" +
67         "   partition_day STRING\n" +
68         ")\n" +
69         "PARTITIONED BY (partition_day) \n" +
70         "WITH (\n" +
71         "   'connector' = 'hudi',\n" +
72         "   'path' = 'file:///D:/flink_hudi_order',\n" +
73         "   'table.type' = 'MERGE_ON_READ',\n" +
74         "   'write.operation' = 'upsert',\n" +
75         "   'hoodie.datasource.write.recordkey.field' = 'orderId',\n" +
76         "   'write.precombine.field' = 'ts',\n" +
77         "   'write.tasks' = '1'\n" +
78         ")\n"
79     );
80
81     tableEnv.executeSql(
82         "INSERT INTO order_hudi_sink\n" +
83         "SELECT\n" +
84         "orderId, userId, orderTime, ip, orderMoney, orderStatus, ts,\n" +
85         "partition_day\n" +
86         "FROM view_order"
87     );
88
89 }
90 }
91

```

## 第五章 利用Hudi进行滴滴数据分析

## SparkUtils

```
1 package com.lwPigKing.hudi.didi
2
3 import org.apache.spark.sql.Session
4
5 /**
6  * Project:   BigDataProject
7  * Create date: 2023/8/7
8  * Created by lwPigKing
9  */
10
11 /**
12  * SparkSQL utility class when manipulating data
13  */
14
15 object SparkUtils {
16
17     def createSparkSession(clazz: Class[_], master: String = "local[4]", partitions: Int = 4):
18     SparkSession = {
19         SparkSession.builder()
20             .appName(clazz.getSimpleName.stripSuffix("$"))
21             .master(master)
22             .config("spark.serializer", "org.apache.spark.serializer.KryoSerializer")
23             .config("spark.sql.shuffle.partitions", partitions)
24             .getOrCreate()
25     }
26 }
27
```

## 分析主要步骤

```
1 package com.lwPigKing.hudi.didi
2
3 import org.apache.spark.sql.functions.{col, concat_ws, unix_timestamp}
4 import org.apache.spark.sql.{DataFrame, SaveMode, Session}
5
6 /**
7  * Project:   BigDataProject
8  * Create date: 2023/8/7
9  * Created by lwPigKing
10  */
11
12 /**
13  * Didi Haikou Mobility operation data analysis uses SparkSQL to manipulate the data,
14  * first read the CSV file, and save it to the Hudi table
15  */
16
17 /**
18  * development major steps
19  * 1. build SparkSession instance objects (integrating Hudi and HDFS)
20  * 2. load the local CSV file Didi trip data
21  * 3. ETL processing
22  * 4. save the converted data to the Hudi table

```

```

23  * 5.stop the SparkSession
24  */
25
26  object DidiStorageSpark {
27      def main(args: Array[String]): Unit = {
28          // dataPath
29          val dataPath: String = "dwv_order_make_haikou_1.txt"
30
31          // Hudi table
32          val hudiTableName: String = "tbl_didi_haikou"
33          val hudiTablePath: String = "/hudi-warehouse/tbl_didi_haikou"
34
35          // step1
36          val sparkSession: SparkSession = SparkUtils.createSparkSession(this.getClass)
37          import sparkSession.implicits._
38
39          // step2
40          val didiDF: DataFrame = sparkSession.read
41              .option("sep", "\\t")
42              .option("header", "true")
43              .option("inferSchema", "true")
44              .csv(dataPath)
45          // didiDF.printSchema()
46          // didiDF.show(10, truncate = false)
47
48          // step3
49          val etlDF: DataFrame = didiDF
50              .withColumn("partitionpath", concat_ws("/", col("year"), col("month"), col("day")))
51              .drop("year", "month", "day")
52              .withColumn("ts", unix_timestamp(col("departure_time"), "yyyy-MM-dd HH:mm:ss"))
53          // etlDF.show(10, truncate = false)
54
55          // step4
56          import org.apache.hudi.DataSourceWriteOptions._
57          import org.apache.hudi.config.HoodieWriteConfig._
58          etlDF.write
59              .mode(SaveMode.Overwrite)
60              .format("hudi")
61              .option("hoodie.insert.shuffle.parallelism", 2)
62              .option("hoodie.upsert.shuffle.parallelism", 2)
63              .option(RECORDKEY_FIELD_OPT_KEY, "order_id")
64              .option(PRECOMBINE_FIELD_OPT_KEY, "ts")
65              .option(PARTITIONPATH_FIELD_OPT_KEY, "partitionpath")
66              .option(TABLE_NAME, hudiTableName)
67              .save(hudiTablePath)
68
69          // step5
70          sparkSession.close()
71
72      }
73  }

```

## Spark分析

```
1 package com.lwPigKing.hudi.didi
2
3 import org.apache.commons.lang3.time.FastDateFormat
4 import org.apache.spark.sql.expressions.UserDefinedFunction
5 import org.apache.spark.sql.functions.{col, sum, udf, when}
6 import org.apache.spark.sql.{DataFrame, SparkSession}
7
8 import java.util.{Calendar, Date}
9
10 /**
11  * Project: BigDataProject
12  * Create date: 2023/8/7
13  * Created by lwPigKing
14  */
15 object DidiAnalysisSpark {
16     def main(args: Array[String]): Unit = {
17         val sparkSession: SparkSession = SparkUtils.createSparkSession(this.getClass, partitions =
18 8)
19
20         import sparkSession.implicits._
21
22         val hudiTablePath: String = "/hudi-warehouse/tbl_didi_haikou"
23         val didiDF: DataFrame = sparkSession.read.format("hudi").load(hudiTablePath)
24         val hudiDF: DataFrame = didiDF.select("order_id", "product_id", "type", "traffic_type",
25 "pre_total_fee", "start_dest_distance", "departure_time"
26 )
27
28         /**
29          * Indicator calculation 1
30          * For the data of Didi Travel in Haikou City,
31          * according to the order type statistics,
32          * the field used: product_id,
33          * the median value [1 Didi car, 2 Didi enterprise car, 3 Didi express, 4 Didi enterprise
34 express]
35          */
36         val reportDF: DataFrame = hudiDF.groupBy("product_id").count()
37         val to_name: UserDefinedFunction = udf(
38             (productID: Int) => {
39                 productID match {
40                     case 1 => "滴滴专车"
41                     case 2 => "滴滴企业专车"
42                     case 3 => "滴滴快车"
43                     case 4 => "滴滴企业快车"
44                 }
45             }
46         )
47
48         val resultDF: DataFrame = reportDF.select(
49             to_name(col("product_id")).as("order_type"),
50             col("count").as("total")
51         )
52
53         resultDF.printSchema()
54         resultDF.show(10, truncate = false)
55
56         /**
57          * Indicator calculation 2
58          * Order timeliness statistics
59          */
60     }
```



```

54      * the filed used: type
55      */
56      val reportDF2: DataFrame = hudiDF.groupBy("type").count()
57      val to_name2: UserDefinedFunction = udf(
58          (realtimeType: Int) => {
59              realtimeType match {
60                  case 0 => "实时"
61                  case 1 => "预约"
62              }
63          }
64      )
65      val resultDF2: DataFrame = reportDF2.select(
66          to_name2(col("type")).as("order_realtime"),
67          col("count").as("total")
68      )
69      reportDF2.printSchema()
70      reportDF2.show(10, truncate = false)
71
72      /**
73       * Indicator calculation 3
74       * Traffic type statistics
75       * the field used: traffic_type
76       */
77      val reportDF3: DataFrame = hudiDF.groupBy("traffic_type").count()
78      val to_name3: UserDefinedFunction = udf(
79          (trafficType: Int) => {
80              trafficType match {
81                  case 0 => "普通散客"
82                  case 1 => "企业时租"
83                  case 2 => "企业接机套餐"
84                  case 3 => "企业送机套餐"
85                  case 4 => "拼车"
86                  case 5 => "接机"
87                  case 6 => "送机"
88                  case 302 => "跨城拼车"
89                  case _ => "未知"
90              }
91          }
92      )
93      val resultDF3: DataFrame = reportDF3.select(
94          to_name3(col("traffic_type")).as("traffic_type"),
95          col("count").as("total")
96      )
97      resultDF3.printSchema()
98      resultDF3.show(10, truncate = false)
99
100     /**
101      * Order price statistics, which will be counted in stages
102      * the field used: pre_total_fee
103      */
104     val resultDF4: DataFrame = hudiDF.agg(
105         // price: 0-15
106         sum(
107             when(
108                 col("pre_total_fee").between(0, 15), 1
109             ).otherwise(0)
110         ).as("0~15"),
111         // price 16-30

```

```

112         sum(
113             when(
114                 col("pre_total_fee").between(16, 30), 1
115             ).otherwise(0)
116         ).as("16~30"),
117         // price: 31-50
118         sum(
119             when(
120                 col("pre_total_fee").between(31, 50), 1
121             ).otherwise(0)
122         ).as("31~50"),
123         // price: 50-100
124         sum(
125             when(
126                 col("pre_total_fee").between(51, 100), 1
127             ).otherwise(0)
128         ).as("51~100"),
129         // price: 100+
130         sum(
131             when(
132                 col("pre_total_fee").gt(100), 1
133             ).otherwise(0)
134         ).as("100+")
135     )
136     resultDF4.printSchema()
137     resultDF4.show(10, truncate = false)
138
139     /**
140     * Order week grouping statistics
141     * the field used: departure_time
142     */
143     val to_week: UserDefinedFunction = udf(
144         (dateStr: String) => {
145             val format: FastDateFormat = FastDateFormat.getInstance("yyyy-MM-dd")
146             val calendar: Calendar = Calendar.getInstance()
147             val date: Date = format.parse(dateStr)
148             calendar.setTime(date)
149             val dayWeek: String = calendar.get(Calendar.DAY_OF_WEEK) match {
150                 case 1 => "星期日"
151                 case 2 => "星期一"
152                 case 3 => "星期二"
153                 case 4 => "星期三"
154                 case 5 => "星期四"
155                 case 6 => "星期五"
156                 case 7 => "星期六"
157             }
158             dayWeek
159         }
160     )
161     val resultDF5: DataFrame = hudiDF.select(
162         to_week(col("departure_time")).as("week")
163     )
164     .groupBy(col("week")).count()
165     .select(
166         col("week"), col("count").as("total")
167     )
168     resultDF5.printSchema()
169     resultDF5.show(10, truncate = false)

```

```
170
171
172     sparkSession.stop()
173
174 }
175 }
176
```