Hudi

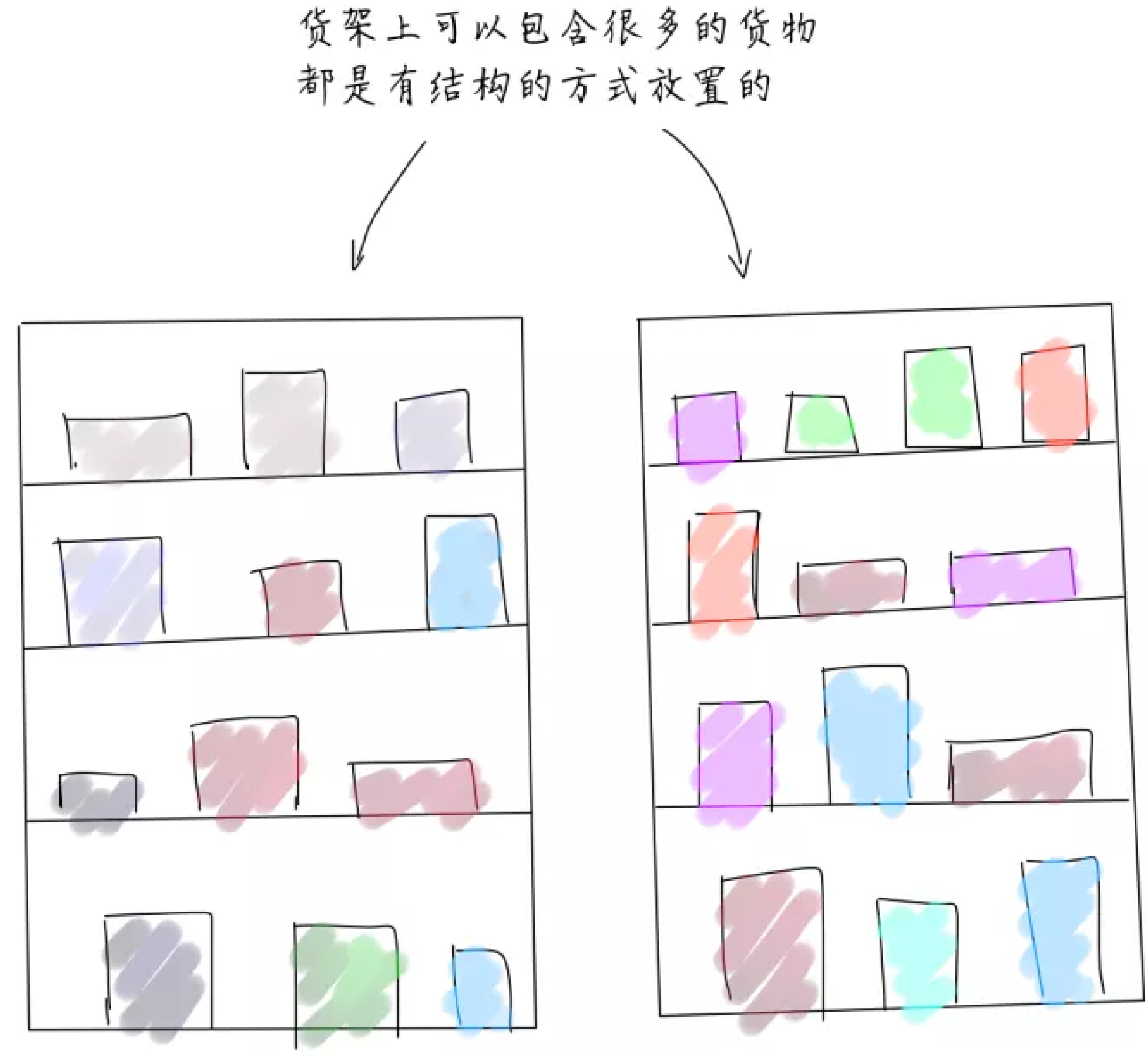
第一章 数据湖的介绍

数据湖Data Lake

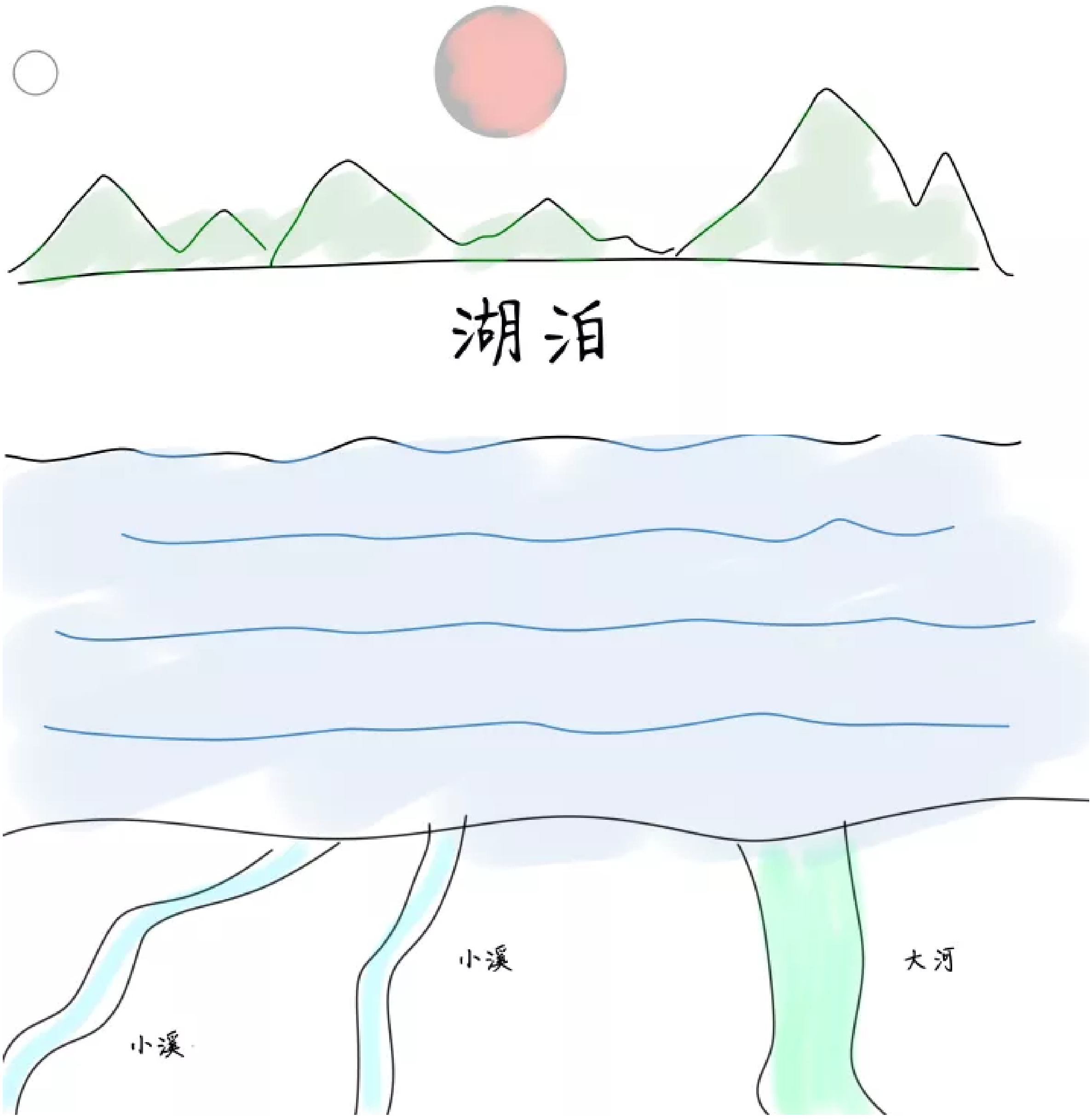
数据湖是大数据架构的新范式，以原始格式存储数据，可以满足用户的广泛需求，并能提供更快的洞察力，细致的数据编录和管理是成功实施数据湖的关键。

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。



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，湖泊可以是由河流、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`。

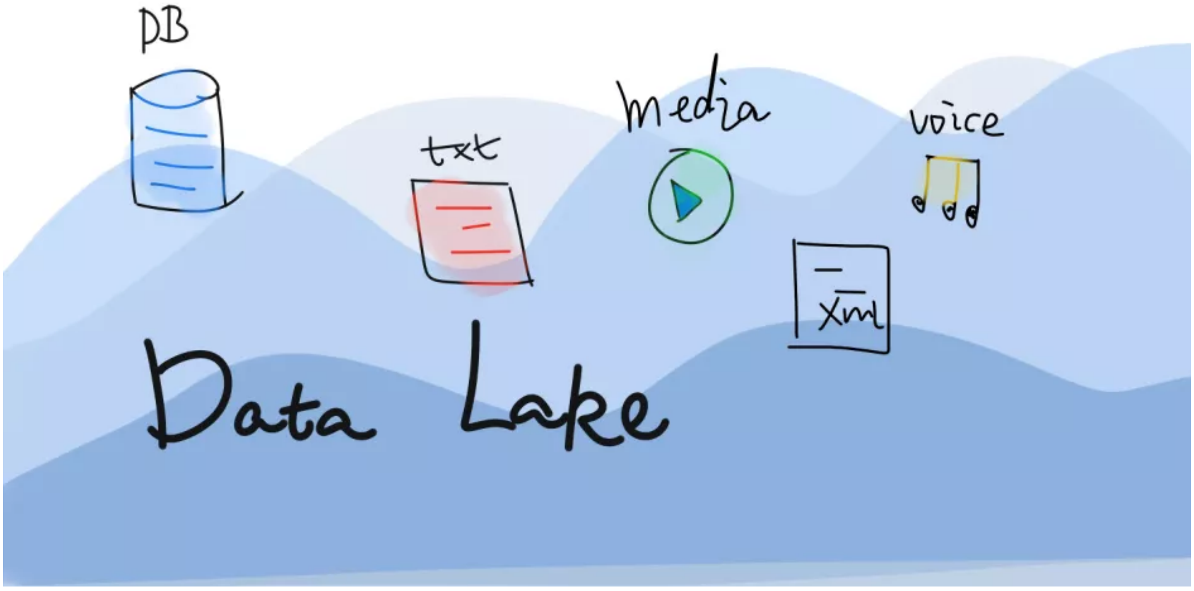


什么是数据湖

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。

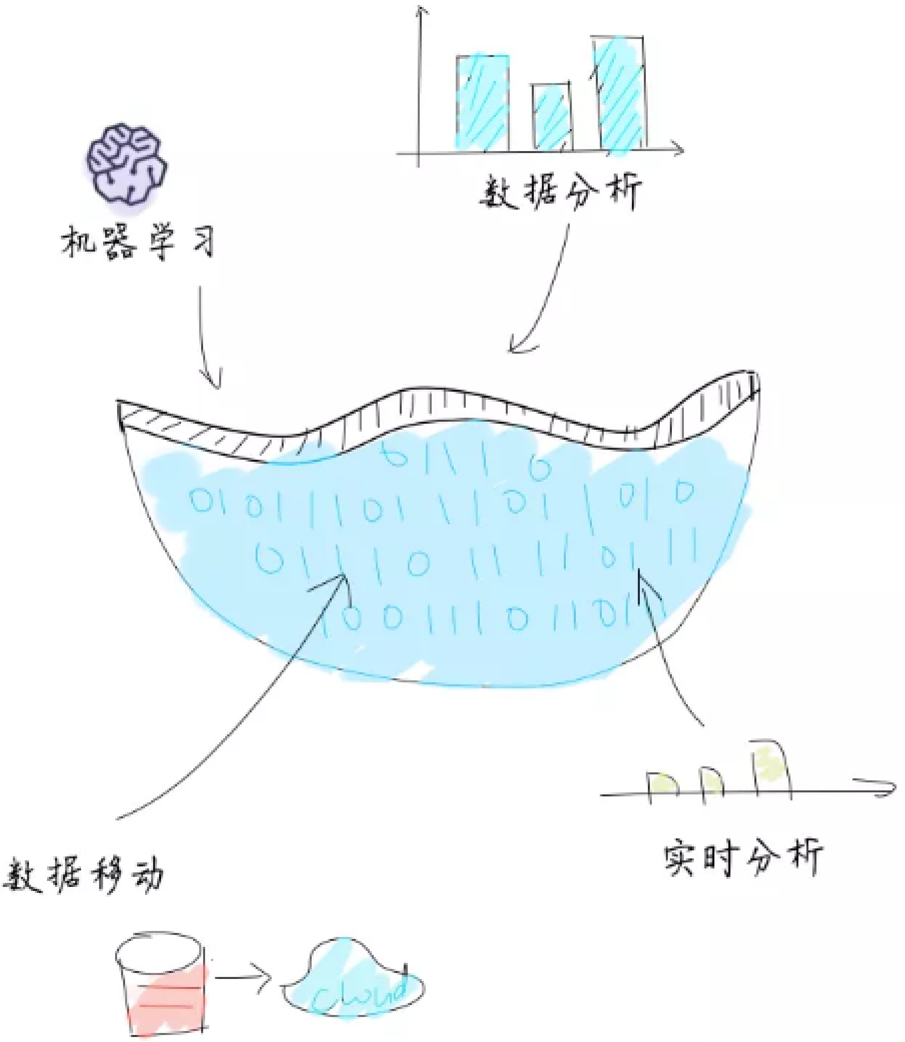
数据湖是一个数据存储库，可以使用数据湖来存储大量的原始数据。现在企业的数据仓库都会通过分层的方式将数据存储在文件夹、文件中，而数据湖使用的是平面架构来存储数据。我们需要做的只是给每个数据元素分配一个唯一的标识符，并通过元数据标签来进行标注。当企业中出现业务问题时，可以从数据湖中查询数据，然后分析业务对应的那一小部分数据集来解决业务问题。



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。



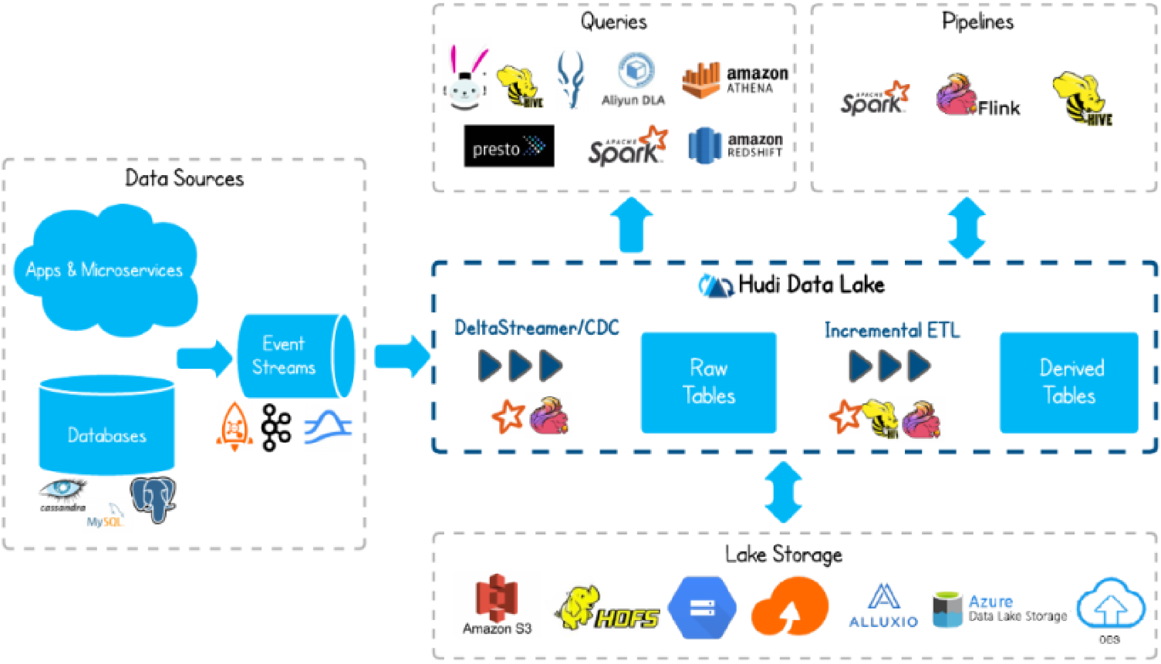
第二章 Hudi概述

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`。

```
package com.lwPigKing.hudi.spark

import org.apache.hudi.QuickstartUtils._
import org.apache.spark.sql.functions.col
import org.apache.spark.sql.{DataFrame, Dataset, Row, SaveMode, SparkSession}

import java.util


/**
 * Project:  BigDataProject
 * Create date:  2023/8/7
 * Created by lwPigKing
 */
object HudiSparkDemo {
   def main(args: Array[String]): Unit = {

      val sparkSession: SparkSession = SparkSession
        .builder()
        .appName(this.getClass.getSimpleName.stripSuffix("$"))
        .master("local[*]")
        .config("spark.serializer", "org.apache.spark.serializer.KryoSerializer")
        .getOrCreate()


      val tableName: String = "tbl_trips_cow"
      val tablePath: String = "/hudi-warehouse/tbl_trips_cow"

      // build data generators that simulate inserting and updating data
      import org.apache.hudi.QuickstartUtils._


      // Task1:Simulate data,insert hudi table and use COW mode
      insertData(sparkSession, tableName, tablePath)

      // Task2:Snapshot Query data in DSL mode
      queryData(sparkSession, tablePath)
      queryDataTime(sparkSession, tablePath)

//     // Task3:Update the data
      val generator: DataGenerator = new DataGenerator()
      insertData(sparkSession, tableName, tablePath, generator)
      updateData(sparkSession, tableName, tablePath, generator)
//
//     // Task4:Incremental Query data in SQL
      incrementalQueryData(sparkSession, tablePath)
//
//     // Task5:Delete the data
      deleteData(sparkSession, tableName, tablePath)


      sparkSession.close()


  }
}


def insertData(sparkSession: SparkSession, table: String, path: String): Unit = {
   import sparkSession.implicits._
   import org.apache.hudi.QuickstartUtils._

   val generator: DataGenerator = new DataGenerator
```

```
    val inserts: util.List[String] = convertToStringList(generator.generateInserts(100))

    import scala.collection.JavaConverters._
    val insertDF: DataFrame = sparkSession
      .read
      .json(sparkSession.sparkContext.parallelize(inserts.asScala, 2).toDS())

    import org.apache.hudi.DataSourceWriteOptions._
    import org.apache.hudi.config.HoodieWriteConfig._
    insertDF.write
      .mode(SaveMode.Append)
      .format("hudi")
      .option("hoodie.insert.shuffle.parallelism", "2")
      .option("hoodie.upsert.shuffle.parallelism", "2")
      .option(PRECOMBINE_FIELD.key(), "ts")
      .option(RECORDKEY_FIELD.key(), "uuid")
      .option(PARTITIONPATH_FIELD.key(), "partitionpath")
      .option(TBL_NAME.key(), table)
      .save(path)

  }


  def queryData(sparkSession: SparkSession, path: String): Unit = {
    import sparkSession.implicits._
    val tripsDF: DataFrame = sparkSession.read.format("hudi").load(path)
    tripsDF.filter(col("fare") >= 20 && col("fare") <= 50)
      .select($"driver", $"rider", $"fare", $"begin_lat", $"begin_log", $"partitionpath", $"_hoodie_commit_time")
      .orderBy($"fare".desc, $"_hoodie_commit_time".desc)
      .show(20, truncate = false)
  }


  def queryDataTime(sparkSession: SparkSession, path: String): Unit = {
    import org.apache.spark.sql.functions._

    // method 1:specify a string in the format yyyyMMddHHmmss
    val df1: Dataset[Row] = sparkSession.read
      .format("hudi")
      .option("as.of.instant", "20211119095057")
      .load(path)
      .sort(col("_hoodie_commit_time").desc)
    df1.show(numRows = 5, truncate = false)

    // method 2:specify a string in the format yyyy-MM-dd HH:mm:ss
    val df2: Dataset[Row] = sparkSession.read
      .format("hudi")
      .option("as.of.instant", "20211119095057")
      .load(path)
      .sort(col("_hoodie_commit_time").desc)
    df2.show(numRows = 5, truncate = false)
  }

  def insertData(sparkSession: SparkSession, table: String, path: String, dataGen: DataGenerator): Unit = {
    import sparkSession.implicits._

```

```scala
    // TODO: a. 模拟乘车数据
    import org.apache.hudi.QuickstartUtils._
    val inserts = convertToStringList(dataGen.generateInserts(100))

    import scala.collection.JavaConverters._
    val insertDF: DataFrame = sparkSession.read
      .json(sparkSession.sparkContext.parallelize(inserts.asScala, 2).toDS())
    //insertDF.printSchema()
    //insertDF.show(10, truncate = false)

    // TODO: b. 插入数据至Hudi表
    import org.apache.hudi.DataSourceWriteOptions._
    import org.apache.hudi.config.HoodieWriteConfig._
    insertDF.write
      .mode(SaveMode.Overwrite)
      .format("hudi") // 指定数据源为Hudi
      .option("hoodie.insert.shuffle.parallelism", "2")
      .option("hoodie.upsert.shuffle.parallelism", "2")
       // Hudi 表的属性设置
      .option(PRECOMBINE_FIELD.key(), "ts")
      .option(RECORDKEY_FIELD.key(), "uuid")
      .option(PARTITIONPATH_FIELD.key(), "partitionpath")
      .option(TBL_NAME.key(), table)
      .save(path)
  }

  def updateData(sparkSession: SparkSession, table: String, path: String, dataGen: DataGenerator): Unit = {
    import sparkSession.implicits._
    import org.apache.hudi.QuickstartUtils._
    import scala.collection.JavaConverters._
    val updates: util.List[String] = convertToStringList(dataGen.generateUpdates(100))
    val updateDF: DataFrame = sparkSession.read
      .json(sparkSession.sparkContext.parallelize(updates.asScala, 2).toDS())

    import org.apache.hudi.DataSourceWriteOptions._
    import org.apache.hudi.config.HoodieWriteConfig._
    updateDF.write
      .mode(SaveMode.Append)
      .format("hudi")
      .option("hoodie.insert.shuffle.parallelism", "2")
      .option("hoodie.upsert.shuffle.parallelism", "2")
      .option(PRECOMBINE_FIELD.key(), "ts")
      .option(RECORDKEY_FIELD.key(), "uuid")
      .option(PARTITIONPATH_FIELD.key(), "partitionpath")
      .option(TBL_NAME.key(), table)
      .save(path)
  }


  def incrementalQueryData(sparkSession: SparkSession, path: String): Unit = {
    import sparkSession.implicits._
    import org.apache.hudi.DataSourceReadOptions._
    sparkSession.read
      .format("hudi")
      .load(path)
      .createOrReplaceTempView("view_temp_hudi_trips")

```

```
   val commits: Array[String] = sparkSession.sql(
      s"""
          |select
          |  distinct(_hoodie_commit_time) as commitTime
          |from
          |  view_temp_hudi_trips
          |order by
          |  commitTime DESC
          |""".stripMargin)
     .map(row => {
         row.getString(0)
     }).take(50)

   val beginTime: String = commits(commits.length - 1)
   println(s"beginTime = ${beginTime}")

   val tripsIncrementalDF: DataFrame = sparkSession.read
     .format("hudi")
     .option(QUERY_TYPE.key(), QUERY_TYPE_INCREMENTAL_OPT_VAL)
     .option(BEGIN_INSTANTTIME.key(), beginTime)
     .load(path)

   tripsIncrementalDF.createOrReplaceTempView("hudi_trips_incremental")
   sparkSession.sql(
      s"""
          |select
          |  _hoodie_commit_time, fare, begin_lon, begin_lat, ts
          |from
          |  hudi_trips_incremental
          |where
          |  fare > 20.0
          |""".stripMargin)
     .show(10, truncate = false)
}


def deleteData(sparkSession: SparkSession, table: String, path: String): Unit = {
   import sparkSession.implicits._
   val tripsDF: DataFrame = sparkSession.read.format("hudi").load(path)
   println(s"Count = ${tripsDF.count()}")

   val value: Dataset[Row] = tripsDF.select($"uuid", $"partitionpath").limit(2)
   import org.apache.hudi.QuickstartUtils._

   val generator: DataGenerator = new DataGenerator()
   val deletes: util.List[String] = generator.generateDeletes(value.collectAsList())

   import scala.collection.JavaConverters._
   val deleteDF: DataFrame =
sparkSession.read.json(sparkSession.sparkContext.parallelize(deletes.asScala, 2))

   import org.apache.hudi.DataSourceWriteOptions._
   import org.apache.hudi.config.HoodieWriteConfig._
   deleteDF.write
     .mode(SaveMode.Append)
     .format("hudi")
     .option("hoodie.insert.shuffle.parallelism", "2")
     .option("hoodie.upsert.shuffle.parallelism", "2")
```

```
        .option(OPERATION.key(), "delete")
        .option(PRECOMBINE_FIELD.key(), "ts")
        .option(RECORDKEY_FIELD.key(), "uuid")
        .option(PARTITIONPATH_FIELD.key(), "partitionpath")
        .option(TBL_NAME.key(), table)
        .save(path)

    val hudiDF: DataFrame = sparkSession.read.format("hudi").load(path)
    println(s"Delete after count = ${hudiDF.count()}")
  }

```

第四章 FlinkSQL开发Hudi

1 读数据

```
package com.lwPigKing.hudi.flink;

import org.apache.flink.table.api.EnvironmentSettings;
import org.apache.flink.table.api.TableEnvironment;

/**
 * Project:  BigDataProject
 * Create date:  2023/8/8
 * Created by lwPigKing
 */
public class FlinkSQLReadDemo {
      public static void main(String[] args) {
            EnvironmentSettings settings = EnvironmentSettings.newInstance().inStreamingMode().build();
            TableEnvironment tableEnv = TableEnvironment.create(settings) ;

            tableEnv.executeSql(
                        "CREATE TABLE order_hudi(\n" +
                                    "  orderId STRING PRIMARY KEY NOT ENFORCED,\n" +
                                    "  userId STRING,\n" +
                                    "  orderTime STRING,\n" +
                                    "  ip STRING,\n" +
                                    "  orderMoney DOUBLE,\n" +
                                    "  orderStatus INT,\n" +
                                    "  ts STRING,\n" +
                                    "  partition_day STRING\n" +
                                    ")\n" +
                                    "PARTITIONED BY (partition_day)\n" +
                                    "WITH (\n" +
                                    "  'connector' = 'hudi',\n" +
                                    "  'path' = 'file:///D:/flink_hudi_order',\n" +
                                    "  'table.type' = 'MERGE_ON_READ',\n" +
                                    "  'read.streaming.enabled' = 'true',\n" +
                                    "  'read.streaming.check-interval' = '4'\n" +
                                    ")"
           );

```

```
        tableEnv.executeSql(
                "SELECT\n" +
                        "orderId, userId, orderTime, ip, orderMoney, orderStatus, ts, partition_day\n" +
                        "FROM order_hudi"
        ).print();

    }
}

```

2 插数据

```
package com.lwPigKing.hudi.flink;

/**
 * Project:  BigDataProject
 * Create date:  2023/8/8
 * Created by lwPigKing
 */

import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
import org.apache.flink.table.api.EnvironmentSettings;
import org.apache.flink.table.api.Table;
import org.apache.flink.table.api.bridge.java.StreamTableEnvironment;

import static org.apache.flink.table.api.Expressions.$;

/**
 * Based on Flink SQL: the data in the topic is consumed in real time,
 * and after conversion processing, it's stored in the Hudi table in real time
 */
public class FlinkSQLHudiDemo {
    public static void main(String[] args) {
        StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
        env.setParallelism(1);
        env.enableCheckpointing(5000);
        EnvironmentSettings settings = EnvironmentSettings.newInstance().inStreamingMode().build();
        StreamTableEnvironment tableEnv = StreamTableEnvironment.create(env, settings);

        tableEnv.executeSql(
                "CREATE TABLE order_kafka_source (\n" +
                        "  orderId STRING,\n" +
                        "  userId STRING,\n" +
                        "  orderTime STRING,\n" +
                        "  ip STRING,\n" +
                        "  orderMoney DOUBLE,\n" +
                        "  orderStatus INT\n" +
                        ") WITH (\n" +
                        "  'connector' = 'kafka',\n" +
                        "  'topic' = 'order-topic',\n" +
                        "  'properties.bootstrap.servers' = 'node1.itcast.cn:9092',\n" +
                        "  'properties.group.id' = 'gid-1001',\n" +
                                    " 'scan.startup.mode' = 'latest-offset',\n" +
                                    " 'format' = 'json',\n" +
                                    " 'json.fail-on-missing-field' = 'false',\n" +
                                    " 'json.ignore-parse-errors' = 'true'\n" +
                                    ")"
        );

        Table etlTable = tableEnv
                    .from("order_kafka_source")
                    .addColumns(
                                    $("orderId").substring(0, 17).as("ts")
                    )
                    .addColumns(
                                    $("orderTime").substring(0, 10).as("partition_day")
                    );
        tableEnv.createTemporaryView("view_order", etlTable);

        tableEnv.executeSql(
                        "CREATE TABLE order_hudi_sink (\n" +
                                    " orderId STRING PRIMARY KEY NOT ENFORCED,\n" +
                                    " userId STRING,\n" +
                                    " orderTime STRING,\n" +
                                    " ip STRING,\n" +
                                    " orderMoney DOUBLE,\n" +
                                    " orderStatus INT,\n" +
                                    " ts STRING,\n" +
                                    " partition_day STRING\n" +
                                    ")\n" +
                                    "PARTITIONED BY (partition_day) \n" +
                                    "WITH (\n" +
                                    " 'connector' = 'hudi',\n" +
                                    " 'path' = 'file:///D:/flink_hudi_order',\n" +
                                    " 'table.type' = 'MERGE_ON_READ',\n" +
                                    " 'write.operation' = 'upsert',\n" +
                                    " 'hoodie.datasource.write.recordkey.field' = 'orderId'," +
                                    " 'write.precombine.field' = 'ts'" +
                                    " 'write.tasks'= '1'" +
                                    ")"
        );

        tableEnv.executeSql(
                        "INSERT INTO order_hudi_sink\n" +
                                    "SELECT\n" +
                                    "orderId, userId, orderTime, ip, orderMoney, orderStatus, ts, partition_day\n" +
                                    "FROM view_order"
        );


    }
}

```

第五章 利用Hudi进行滴滴数据分析

SparkUtils

```scala
package com.lwPigKing.hudi.didi

import org.apache.spark.sql.SparkSession

/**
 * Project:  BigDataProject
 * Create date:  2023/8/7
 * Created by lwPigKing
 */

/**
 * SparkSQL utility class when manipulating data
 */

object SparkUtils {

  def createSparkSession(clazz: Class[_], master: String = "local[4]", partitions: Int = 4): SparkSession = {
      SparkSession.builder()
        .appName(clazz.getSimpleName.stripSuffix("$"))
        .master(master)
        .config("spark.serializer", "org.apache.spark.serializer.KryoSerializer")
        .config("spark.sql.shuffle.partitions", partitions)
        .getOrCreate()
  }

}

```

分析主要步骤

```scala
package com.lwPigKing.hudi.didi

import org.apache.spark.sql.functions.{col, concat_ws, unix_timestamp}
import org.apache.spark.sql.{DataFrame, SaveMode, SparkSession}

/**
 * Project:  BigDataProject
 * Create date:  2023/8/7
 * Created by lwPigKing
 */

/**
 * Didi Haikou Mobility operation data analysis uses SparkSQL to manipulate the data,
 * first read the CSV file,and save it to the Hudi table
 */

/**
 * development major steps
 * 1.build SparkSession instance objects(integrating Hudi and HDFS)
 * 2.load the local CSV file Didi trip data
 * 3.ETL processing
 * 4.save the converted data to the Hudi table
 * 5.stop the SparkSession
 */

object DidiStorageSpark {
  def main(args: Array[String]): Unit = {
    // dataPath
    val dataPath: String = "dwv_order_make_haikou_1.txt"

    // Hudi table
    val hudiTableName: String = "tbl_didi_haikou"
    val hudiTablePath: String = "/hudi-warehouse/tbl_didi_haikou"

    // step1
    val sparkSession: SparkSession = SparkUtils.createSparkSession(this.getClass)
    import sparkSession.implicits._

    // step2
    val didiDF: DataFrame = sparkSession.read
      .option("sep", "\\t")
      .option("header", "true")
      .option("inferSchema", "true")
      .csv(dataPath)
//     didiDF.printSchema()
//     didiDF.show(10, truncate = false)

    // step3
    val etlDF: DataFrame = didiDF
      .withColumn("partitionpath", concat_ws("/", col("year"), col("month"), col("day")))
      .drop("year", "month", "day")
      .withColumn("ts", unix_timestamp(col("departure_time"), "yyyy-MM-dd HH:mm:ss"))
//     etlDF.show(10, truncate = false)

    // step4
    import org.apache.hudi.DataSourceWriteOptions._
    import org.apache.hudi.config.HoodieWriteConfig._
    etlDF.write
      .mode(SaveMode.Overwrite)
      .format("hudi")
      .option("hoodie.insert.shuffle.parallelism", 2)
      .option("hoodie.upsert.shuffle.parallelism", 2)
      .option(RECORDKEY_FIELD_OPT_KEY, "order_id")
      .option(PRECOMBINE_FIELD_OPT_KEY, "ts")
      .option(PARTITIONPATH_FIELD_OPT_KEY, "partitionpath")
      .option(TABLE_NAME, hudiTableName)
      .save(hudiTablePath)

    // step5
    sparkSession.close()

  }
}
```

Spark分析

```scala
package com.lwPigKing.hudi.didi

import org.apache.commons.lang3.time.FastDateFormat
import org.apache.spark.sql.expressions.UserDefinedFunction
import org.apache.spark.sql.functions.{col, sum, udf, when}
import org.apache.spark.sql.{DataFrame, SparkSession}

import java.util.{Calendar, Date}

/**
 * Project:  BigDataProject
 * Create date:  2023/8/7
 * Created by lwPigKing
 */
object DidiAnalysisSpark {
   def main(args: Array[String]): Unit = {
      val sparkSession: SparkSession = SparkUtils.createSparkSession(this.getClass, partitions = 8)
      import sparkSession.implicits._

      val hudiTablePath: String = "/hudi-warehouse/tbl_didi_haikou"
      val didiDF: DataFrame = sparkSession.read.format("hudi").load(hudiTablePath)
      val hudiDF: DataFrame = didiDF.select("order_id", "product_id", "type", "traffic_type",
         "pre_total_fee", "start_dest_distance", "departure_time"
     )

      /**
       * Indicator calculation 1
       * For the data of Didi Travel in Haikou City,
       * according to the order type statistics,
       * the field used: product_id,
       * the median value [1 Didi car, 2 Didi enterprise car, 3 Didi express, 4 Didi enterprise express]
       */
      val reportDF: DataFrame = hudiDF.groupBy("product_id").count()
      val to_name: UserDefinedFunction = udf(
        (productID: Int) => {
            productID match {
               case 1 => "滴滴专车"
               case 2 => "滴滴企业专车"
               case 3 => "滴滴快车"
               case 4 => "滴滴企业快车"
           }
        }
     )
      val resultDF: DataFrame = reportDF.select(
         to_name(col("product_id")).as("order_type"),
         col("count").as("total")
     )
      resultDF.printSchema()
      resultDF.show(10, truncate = false)

      /**
       * Indicator calculation 2
       * Order timeliness statistics
      * the filed used: type
      */
     val reportDF2: DataFrame = hudiDF.groupBy("type").count()
     val to_name2: UserDefinedFunction = udf(
       (realtimeType: Int) => {
           realtimeType match {
              case 0 => "实时"
              case 1 => "预约"
          }
       }
    )
     val resultDF2: DataFrame = reportDF2.select(
        to_name2(col("type")).as("order_realtime"),
        col("count").as("total")
    )
     reportDF2.printSchema()
     reportDF2.show(10, truncate = false)

     /**
      * Indicator calculation 3
      * Traffic type statistics
      * the field used: traffic_type
      */
     val reportDF3: DataFrame = hudiDF.groupBy("traffic_type").count()
     val to_name3: UserDefinedFunction = udf(
       (trafficType: Int) => {
           trafficType match {
              case 0 => "普通散客"
              case 1 => "企业时租"
              case 2 => "企业接机套餐"
              case 3 => "企业送机套餐"
              case 4 => "拼车"
              case 5 => "接机"
              case 6 => "送机"
              case 302 => "跨城拼车"
              case _ => "未知"
          }
       }
    )
     val resultDF3: DataFrame = reportDF3.select(
        to_name3(col("traffic_type")).as("traffic_type"),
        col("count").as("total")
    )
     resultDF3.printSchema()
     resultDF3.show(10, truncate = false)

     /**
      * Order price statistics, which will be counted in stages
      * the field used: pre_total_fee
      */
     val resultDF4: DataFrame = hudiDF.agg(
        // price: 0-15
        sum(
           when(
              col("pre_total_fee").between(0, 15), 1
          ).otherwise(0)
       ).as("0~15"),
        // price 16-30
```

```
           sum(
              when(
                 col("pre_total_fee").between(16, 30), 1
             ).otherwise(0)
          ).as("16~30"),
           // price: 31-50
           sum(
              when(
                 col("pre_total_fee").between(31, 50), 1
             ).otherwise(0)
          ).as("31~50"),
           // price: 50-100
           sum(
              when(
                 col("pre_total_fee").between(51, 100), 1
             ).otherwise(0)
          ).as("51~100"),
           // price: 100+
           sum(
              when(
                 col("pre_total_fee").gt(100), 1
             ).otherwise(0)
          ).as("100+")
        )
        resultDF4.printSchema()
        resultDF4.show(10, truncate = false)

        /**
         * Order week grouping statistics
         * the field used: departure_time
         */
        val to_week: UserDefinedFunction = udf(
          (dateStr: String) => {
              val format: FastDateFormat = FastDateFormat.getInstance("yyyy-MM-dd")
              val calendar: Calendar = Calendar.getInstance()
              val date: Date = format.parse(dateStr)
              calendar.setTime(date)
              val dayWeek: String = calendar.get(Calendar.DAY_OF_WEEK) match {
                 case 1 => "星期日"
                 case 2 => "星期一"
                 case 3 => "星期二"
                 case 4 => "星期三"
                 case 5 => "星期四"
                 case 6 => "星期五"
                 case 7 => "星期六"
             }
              dayWeek
          }
        )
        val resultDF5: DataFrame = hudiDF.select(
           to_week(col("departure_time")).as("week")
        )
          .groupBy(col("week")).count()
          .select(
              col("week"), col("count").as("total")
          )
        resultDF5.printSchema()
        resultDF5.show(10, truncate = false)
```

```


      sparkSession.stop()

  }
}

```