Apache Spark SQL 参数介绍

我们可以在初始化 SparkSession 的时候进行一些设置：

```
import org.apache.spark.sql.SparkSession
val spark: SparkSession = SparkSession.builder
  .master("local[*]")
  .appName("My Spark Application")
  .config("spark.sql.warehouse.dir", "c:/Temp") (1)
  .getOrCreate
Sets spark.sql.warehouse.dir for the Spark SQL session
```

也可以使用 SQL SET 命令进行设置：

```
scala> spark.conf.getOption("spark.sql.hive.metastore.version")
res1: Option[String] = None

scala> spark.sql("SET spark.sql.hive.metastore.version=2.3.2").show(truncate = false)
+--------------------------------+-----+
|key                             |value|
+--------------------------------+-----+
|spark.sql.hive.metastore.version|2.3.2|
+--------------------------------+-----+

scala> spark.conf.get("spark.sql.hive.metastore.version")
res2: String = 2.3.2
```

其他方式的设置可以参见 Spark三种属性配置方式详细说明。下面是 Spark SQL 支持的参数列表。

spark.sql.adaptive.enabled

Enables adaptive query execution

Default: false

Use SQLConf.adaptiveExecutionEnabled method to access the current value.

spark.sql.adaptive.minNumPostShufflePartitions

(internal) The advisory minimal number of post-shuffle partitions for ExchangeCoordinator.

Default: -1

This setting is used in Spark SQL tests to have enough parallelism to expose issues that will not be exposed with a single partition. Only positive values are used.

Use SQLConf.minNumPostShufflePartitions method to access the current value.

spark.sql.adaptive.shuffle.targetPostShuffleInputSize

Recommended size of the input data of a post-shuffle partition (in bytes)

Default: 64 * 1024 * 1024 bytes

Use SQLConf.targetPostShuffleInputSize method to access the current value.

spark.sql.allowMultipleContexts

Controls whether creating multiple SQLContexts/HiveContexts is allowed (true) or not (false)

Default: true

spark.sql.autoBroadcastJoinThreshold

Maximum size (in bytes) for a table that will be broadcast to all worker nodes when performing a join.

Default: 10L * 1024 * 1024 (10M)

If the size of the statistics of the logical plan of a table is at most the setting, the DataFrame is broadcast for join.

Negative values or 0 disable broadcasting.

Use SQLConf.autoBroadcastJoinThreshold method to access the current value.

spark.sql.avro.compression.codec

The compression codec to use when writing Avro data to disk

Default: snappy

The supported codecs are:

- uncompressed
- deflate
- snappy
- bzip2
- xz

Use SQLConf.avroCompressionCodec method to access the current value.

spark.sql.broadcastTimeout

Timeout in seconds for the broadcast wait time in broadcast joins.

Default: 5 * 60

When negative, it is assumed infinite (i.e. Duration.Inf)

Use SQLConf.broadcastTimeout method to access the current value.

spark.sql.caseSensitive

(internal) Controls whether the query analyzer should be case sensitive (true) or not (false).

Default: false

It is highly discouraged to turn on case sensitive mode.

Use SQLConf.caseSensitiveAnalysis method to access the current value.

spark.sql.cbo.enabled

Enables cost-based optimization (CBO) for estimation of plan statistics when true.

Default: false

Use SQLConf.cboEnabled method to access the current value.

spark.sql.cbo.joinReorder.enabled

Enables join reorder for cost-based optimization (CBO).

Default: false

Use SQLConf.joinReorderEnabled method to access the current value.

spark.sql.cbo.starSchemaDetection

Enables join reordering based on star schema detection for cost-based optimization (CBO) in ReorderJoin logical plan optimization.

Default: false

Use SQLConf.starSchemaDetection method to access the current value.

spark.sql.codegen.comments

Controls whether CodegenContext should register comments (true) or not (false).

Default: false

spark.sql.codegen.factoryMode

(internal) Determines the codegen generator fallback behavior

Default: FALLBACK

Acceptable values:

- CODEGEN_ONLY - disable fallback mode
- FALLBACK - try codegen first and, if any compile error happens, fallback to interpreted mode
- NO_CODEGEN - skips codegen and always uses interpreted path

Used when CodeGeneratorWithInterpretedFallback is requested to createObject (when UnsafeProjection is requested to create an UnsafeProjection for Catalyst expressions)

spark.sql.codegen.fallback

(internal) Whether the whole stage codegen could be temporary disabled for the part of a query that has failed to compile generated code (true) or not (false).

Default: true

Use SQLConf.wholeStageFallback method to access the current value.

spark.sql.codegen.hugeMethodLimit

(internal) The maximum bytecode size of a single compiled Java function generated by whole-stage codegen.

Default: 65535

The default value 65535 is the largest bytecode size possible for a valid Java method. When running on HotSpot, it may be preferable to set the value to 8000 (which is the value of HugeMethodLimit in the OpenJDK JVM settings)

Use SQLConf.hugeMethodLimit method to access the current value.

spark.sql.codegen.useIdInClassName

(internal) Controls whether to embed the (whole-stage) codegen stage ID into the class name of the generated class as a suffix (true) or not (false)

Default: true

Use SQLConf.wholeStageUseIdInClassName method to access the current value.

spark.sql.codegen.maxFields

(internal) Maximum number of output fields (including nested fields) that whole-stage codegen supports. Going above the number deactivates whole-stage codegen.

Default: 100

Use SQLConf.wholeStageMaxNumFields method to access the current value.

spark.sql.codegen.splitConsumeFuncByOperator

(internal) Controls whether whole stage codegen puts the logic of consuming rows of each physical operator into individual methods, instead of a single big method. This can be used to avoid oversized function that can miss the opportunity of JIT optimization.

Default: true

Use SQLConf.wholeStageSplitConsumeFuncByOperator method to access the current value.

spark.sql.codegen.wholeStage

(internal) Whether the whole stage (of multiple physical operators) will be compiled into a single Java method (true) or not (false).

Default: true

Use SQLConf.wholeStageEnabled method to access the current value.

spark.sql.columnVector.offheap.enabled

(internal) Enables OffHeapColumnVector in ColumnarBatch (true) or not (false). When false, OnHeapColumnVector is used instead.

Default: false

Use SQLConf.offHeapColumnVectorEnabled method to access the current value.

spark.sql.columnNameOfCorruptRecord

spark.sql.constraintPropagation.enabled

(internal) When true, the query optimizer will infer and propagate data constraints in the query plan to optimize them. Constraint propagation can sometimes be computationally expensive for certain kinds of query plans (such as those with a large number of predicates and aliases) which might negatively impact overall runtime.

Default: true

Use SQLConf.constraintPropagationEnabled method to access the current value.

spark.sql.defaultSizeInBytes

(internal) Estimated size of a table or relation used in query planning

Default: Java's Long.MaxValue

Set to Java's Long.MaxValue which is larger than spark.sql.autoBroadcastJoinThreshold to be more conservative. That is to say by default the optimizer will not choose to broadcast a table unless it knows for sure that the table size is small enough.

Used by the planner to decide when it is safe to broadcast a relation. By default, the system will assume that tables are too large to broadcast.

Use SQLConf.defaultSizeInBytes method to access the current value.

spark.sql.dialect

spark.sql.exchange.reuse

(internal) When enabled (i.e. true), the Spark planner will find duplicated exchanges and subqueries and re-use them.

Default: true

Note
When disabled (i.e. false), ReuseSubquery and ReuseExchange physical optimizations (that the Spark planner uses for physical query plan optimization) do nothing.
Use SQLConf.exchangeReuseEnabled method to access the current value.

spark.sql.execution.useObjectHashAggregateExec

Enables ObjectHashAggregateExec when Aggregation execution planning strategy is executed.

Default: true

Use SQLConf.useObjectHashAggregation method to access the current value.

spark.sql.files.ignoreCorruptFiles

Controls whether to ignore corrupt files (true) or not (false). If true, the Spark jobs will continue to run when encountering corrupted files and the contents that have been read will still be returned.

Default: false

Use SQLConf.ignoreCorruptFiles method to access the current value.

spark.sql.files.ignoreMissingFiles

Controls whether to ignore missing files (true) or not (false). If true, the Spark jobs will continue to run when encountering missing files and the contents that have been read will still be returned.

Default: false

Use SQLConf.ignoreMissingFiles method to access the current value.

spark.sql.files.maxPartitionBytes

The maximum number of bytes to pack into a single partition when reading files.

Default: 128 * 1024 * 1024 (which corresponds to parquet.block.size)

Use SQLConf.filesMaxPartitionBytes method to access the current value.

spark.sql.files.openCostInBytes

(internal) The estimated cost to open a file, measured by the number of bytes could be scanned at the same time (to include multiple files into a partition).

Default: 4 * 1024 * 1024

It’s better to over estimate it, then the partitions with small files will be faster than partitions with bigger files (which is scheduled first).

Use SQLConf.filesOpenCostInBytes method to access the current value.

spark.sql.files.maxRecordsPerFile

Maximum number of records to write out to a single file. If this value is 0 or negative, there is no limit.

Default: 0

Use SQLConf.maxRecordsPerFile method to access the current value.

spark.sql.inMemoryColumnarStorage.batchSize

(internal) Controls...FIXME

Default: 10000

Use SQLConf.columnBatchSize method to access the current value.

spark.sql.inMemoryColumnarStorage.compressed

(internal) Controls...FIXME

Default: true

Use SQLConf.useCompression method to access the current value.

spark.sql.inMemoryColumnarStorage.enableVectorizedReader

Enables vectorized reader for columnar caching.

Default: true

Use SQLConf.cacheVectorizedReaderEnabled method to access the current value.

spark.sql.inMemoryColumnarStorage.partitionPruning

(internal) Enables partition pruning for in-memory columnar tables

Default: true

Use SQLConf.inMemoryPartitionPruning method to access the current value.

spark.sql.join.preferSortMergeJoin

(internal) Controls whether JoinSelection execution planning strategy prefers sort merge join over shuffled hash join.

Default: true

Use SQLConf.preferSortMergeJoin method to access the current value.

spark.sql.legacy.rdd.applyConf

(internal) Enables propagation of SQL configurations when executing operations on the RDD that represents a structured query. This is the (buggy) behavior up to 2.4.4.

Default: true

This is for cases not tracked by SQL execution, when a Dataset is converted to an RDD either using rdd operation or QueryExecution, and then the returned RDD is used to invoke actions on it.

This config is deprecated and will be removed in 3.0.0.

spark.sql.legacy.replaceDatabricksSparkAvro.enabled

Enables resolving (mapping) the data source provider com.databricks.spark.avro to the built-in (but external) Avro data source module for backward compatibility.

Default: true

Use SQLConf.replaceDatabricksSparkAvroEnabled method to access the current value.

spark.sql.limit.scaleUpFactor

(internal) Minimal increase rate in the number of partitions between attempts when executing take operator on a structured query. Higher values lead to more partitions read. Lower values might lead to longer execution times as more jobs will be run.

Default: 4

Use SQLConf.limitScaleUpFactor method to access the current value.

spark.sql.optimizer.excludedRules

Comma-separated list of optimization rule names that should be disabled (excluded) in the optimizer. The optimizer will log the rules that have indeed been excluded.

Default: (empty)

Note
It is not guaranteed that all the rules in this configuration will eventually be excluded, as some rules are necessary for correctness.
Use SQLConf.optimizerExcludedRules method to access the current value.

spark.sql.optimizer.inSetConversionThreshold

(internal) The threshold of set size for InSet conversion.

Default: 10

Use SQLConf.optimizerInSetConversionThreshold method to access the current value.

spark.sql.optimizer.maxIterations

Maximum number of iterations for Analyzer and Optimizer.

Default: 100

spark.sql.optimizer.replaceExceptWithFilter

(internal) When true, the apply function of the rule verifies whether the right node of the except operation is of type Filter or Project followed by Filter. If yes, the rule further verifies 1) Excluding the filter operations from the right (as well as the left node, if any) on the top, whether both the nodes evaluates to a same result. 2) The left and right nodes don't contain any SubqueryExpressions. 3) The output column names of the left node are distinct. If all the conditions are met, the rule will replace the except operation with a Filter by flipping the filter condition(s) of the right node.

Default: true

spark.sql.orc.impl

(internal) When native, use the native version of ORC support instead of the ORC library in Hive 1.2.1.

Default: native

Acceptable values:

hive

native

spark.sql.parquet.binaryAsString

Some other Parquet-producing systems, in particular Impala and older versions of Spark SQL, do not differentiate between binary data and strings when writing out the Parquet schema. This flag tells Spark SQL to interpret binary data as a string to provide compatibility with these systems.

Default: false

Use SQLConf.isParquetBinaryAsString method to access the current value.

spark.sql.parquet.columnarReaderBatchSize

The number of rows to include in a parquet vectorized reader batch (the capacity of VectorizedParquetRecordReader).

Default: 4096 (4k)

The number should be carefully chosen to minimize overhead and avoid OOMs while reading data.

Use SQLConf.parquetVectorizedReaderBatchSize method to access the current value.

spark.sql.parquet.int96AsTimestamp

Some Parquet-producing systems, in particular Impala, store Timestamp into INT96. Spark would also store Timestamp as INT96 because we need to avoid precision lost of the nanoseconds field. This flag tells Spark SQL to interpret INT96 data as a timestamp to provide compatibility with these systems.

Default: true

Use SQLConf.isParquetINT96AsTimestamp method to access the current value.

spark.sql.parquet.enableVectorizedReader

Enables vectorized parquet decoding.

Default: true

Use SQLConf.parquetVectorizedReaderEnabled method to access the current value.

spark.sql.parquet.filterPushdown

Controls the filter predicate push-down optimization for data sources using parquet file format

Default: true

Use SQLConf.parquetFilterPushDown method to access the current value.

spark.sql.parquet.filterPushdown.date

(internal) Enables parquet filter push-down optimization for Date (when spark.sql.parquet.filterPushdown is enabled)

Default: true

Use SQLConf.parquetFilterPushDownDate method to access the current value.

spark.sql.parquet.int96TimestampConversion

Controls whether timestamp adjustments should be applied to INT96 data when converting to timestamps, for data written by Impala.

Default: false

This is necessary because Impala stores INT96 data with a different timezone offset than Hive and Spark.

Use SQLConf.isParquetINT96TimestampConversion method to access the current value.

spark.sql.parquet.recordLevelFilter.enabled

Enables Parquet’s native record-level filtering using the pushed down filters.

Default: false

Note
This configuration only has an effect when spark.sql.parquet.filterPushdown is enabled (and it is by default).
Use SQLConf.parquetRecordFilterEnabled method to access the current value.

spark.sql.parser.quotedRegexColumnNames

Controls whether quoted identifiers (using backticks) in SELECT statements should be interpreted as regular expressions.

Default: false

Use SQLConf.supportQuotedRegexColumnName method to access the current value.

spark.sql.sort.enableRadixSort

(internal) Controls whether to use radix sort (true) or not (false) in ShuffleExchangeExec and SortExec physical operators

Default: true

Radix sort is much faster but requires additional memory to be reserved up-front. The memory overhead may be significant when sorting very small rows (up to 50% more).

Use SQLConf.enableRadixSort method to access the current value.

spark.sql.sources.commitProtocolClass

(internal) Fully-qualified class name of the FileCommitProtocol to use for…FIXME

Default: SQLHadoopMapReduceCommitProtocol

Use SQLConf.fileCommitProtocolClass method to access the current value.

spark.sql.sources.partitionOverwriteMode

Enables dynamic partition inserts when dynamic

Default: static

When INSERT OVERWRITE a partitioned data source table with dynamic partition columns, Spark SQL supports two modes (case-insensitive):

static - Spark deletes all the partitions that match the partition specification (e.g. PARTITION(a=1,b)) in the INSERT statement, before overwriting

dynamic - Spark doesn’t delete partitions ahead, and only overwrites those partitions that have data written into it

The default (STATIC) is to keep the same behavior of Spark prior to 2.3. Note that this config doesn’t affect Hive serde tables, as they are always overwritten with dynamic mode.

Use SQLConf.partitionOverwriteMode method to access the current value.

spark.sql.pivotMaxValues

Maximum number of (distinct) values that will be collected without error (when doing a pivot without specifying the values for the pivot column)

Default: 10000

Use SQLConf.dataFramePivotMaxValues method to access the current value.

spark.sql.redaction.options.regex

Regular expression to find options of a Spark SQL command with sensitive information

Default: (?i)secret!password

The values of the options matched will be redacted in the explain output.

This redaction is applied on top of the global redaction configuration defined by spark.redaction.regex configuration.

Used exclusively when SQLConf is requested to redactOptions.

spark.sql.redaction.string.regex

Regular expression to point at sensitive information in text output

Default: (undefined)

When this regex matches a string part, it is replaced by a dummy value (i.e. ***(redacted)). This is currently used to redact the output of SQL explain commands.

Note
When this conf is not set, the value of spark.redaction.string.regex is used instead.
Use SQLConf.stringRedactionPattern method to access the current value.

spark.sql.retainGroupColumns

Controls whether to retain columns used for aggregation or not (in RelationalGroupedDataset operators).

Default: true

Use SQLConf.dataFrameRetainGroupColumns method to access the current value.

spark.sql.runSQLOnFiles

(internal) Controls whether Spark SQL could use datasource.path as a table in a SQL query.

Default: true

Use SQLConf.runSQLonFile method to access the current value.

spark.sql.selfJoinAutoResolveAmbiguity

Controls whether to resolve ambiguity in join conditions for self-joins automatically (true) or not (false)

Default: true

spark.sql.session.timeZone

The ID of session-local timezone, e.g. "GMT", "America/Los_Angeles", etc.

Default: Java's TimeZone.getDefault.getID

Use SQLConf.sessionLocalTimeZone method to access the current value.

spark.sql.shuffle.partitions

Number of partitions to use by default when shuffling data for joins or aggregations

Default: 200

Corresponds to Apache Hive's mapred.reduce.tasks property that Spark considers deprecated.

Use SQLConf.numShufflePartitions method to access the current value.

spark.sql.sources.bucketing.enabled

Enables bucketing support. When disabled (i.e. false), bucketed tables are considered regular (non-bucketed) tables.

Default: true

Use SQLConf.bucketingEnabled method to access the current value.

spark.sql.sources.default

Defines the default data source to use for DataFrameReader.

Default: parquet

Used when:

Reading (DataFrameWriter) or writing (DataFrameReader) datasets

Creating external table from a path (in Catalog.createExternalTable)

Reading (DataStreamReader) or writing (DataStreamWriter) in Structured Streaming

spark.sql.statistics.fallBackToHdfs

Enables automatic calculation of table size statistic by falling back to HDFS if the table statistics are not available from table metadata.

Default: false

This can be useful in determining if a table is small enough for auto broadcast joins in query planning.

Use SQLConf.fallBackToHdfsForStatsEnabled method to access the current value.

spark.sql.statistics.histogram.enabled

Enables generating histograms when computing column statistics

Default: false

Note
Histograms can provide better estimation accuracy. Currently, Spark only supports equi-height histogram. Note that collecting histograms takes extra cost. For example, collecting column statistics usually takes only one table scan, but generating equi-height histogram will cause an extra table scan.
Use SQLConf.histogramEnabled method to access the current value.

spark.sql.statistics.histogram.numBins

(internal) The number of bins when generating histograms.

Default: 254

Note
The number of bins must be greater than 1.
Use SQLConf.histogramNumBins method to access the current value.

spark.sql.statistics.parallelFileListingInStatsComputation.enabled

(internal) Enables parallel file listing in SQL commands, e.g. ANALYZE TABLE (as opposed to single thread listing that can be particularly slow with tables with hundreds of partitions)

Default: true

Use SQLConf.parallelFileListingInStatsComputation method to access the current value.

spark.sql.statistics.ndv.maxError

(internal) The maximum estimation error allowed in HyperLogLog++ algorithm when generating column level statistics.

Default: 0.05

spark.sql.statistics.percentile.accuracy

(internal) Accuracy of percentile approximation when generating equi-height histograms. Larger value means better accuracy. The relative error can be deduced by 1.0 / PERCENTILE_ACCURACY.

Default: 10000

spark.sql.statistics.size.autoUpdate.enabled

Enables automatic update of the table size statistic of a table after the table has changed.

Default: false

Important
If the total number of files of the table is very large this can be expensive and slow down data change commands.
Use SQLConf.autoSizeUpdateEnabled method to access the current value.

spark.sql.subexpressionElimination.enabled

(internal) Enables subexpression elimination

Default: true

Use subexpressionEliminationEnabled method to access the current value.

spark.sql.truncateTable.ignorePermissionAcl.enabled

(internal) Disables setting back original permission and ACLs when re-creating the table/partition paths for TRUNCATE TABLE command.

Default: false

Use truncateTableIgnorePermissionAcl method to access the current value.

spark.sql.ui.retainedExecutions

The number of SQLExecutionUIData entries to keep in failedExecutions and completedExecutions internal registries.

Default: 1000

When a query execution finishes, the execution is removed from the internal activeExecutions registry and stored in failedExecutions or completedExecutions given the end execution status. It is when SQLListener makes sure that the number of SQLExecutionUIData entires does not exceed spark.sql.ui.retainedExecutions Spark property and removes the excess of entries.

spark.sql.windowExec.buffer.in.memory.threshold

(internal) Threshold for number of rows guaranteed to be held in memory by WindowExec physical operator.

Default: 4096

Use windowExecBufferInMemoryThreshold method to access the current value.

spark.sql.windowExec.buffer.spill.threshold

(internal) Threshold for number of rows buffered in a WindowExec physical operator.

Default: 4096

Use windowExecBufferSpillThreshold method to access the current value.