Flink 1.11 Release 文档

集群和部署

- **支持 Hadoop 3.0 及更高的版本**：Flink 不再提供任何 `flink-shaded-hadoop-` 依赖。用户可以通过配置 HADOOP_CLASSPATH 环境变量(推荐)或在 lib 文件夹下放入 Hadoop 依赖项。另外 `include-hadoop` Maven profile 也已经被移除了。
- **移除了 LegacyScheduler**：Flink 不再支持 legacy scheduler，如果你设置了 `jobmanager.scheduler: legacy` 将不再起作用并且会抛出 IllegalArgumentException 异常，该参数的默认值并且是唯一选项为 ng。
- **将用户代码的类加载器和 slot 的生命周期进行绑定**：只要为单个作业分配了至少一个 slot，TaskManager 就会重新使用用户代码的类加载器。这会稍微改变 Flink 的恢复行为，从而不会重新加载静态字段。这样做的好处是，可以大大减轻对 JVM metaspace 的压力。
- **slave 文件重命名为 workers**：对于 Standalone 模式安装，worker 节点文件不再是 slaves 而是 workers，以前使用 `start-cluster.sh` 和 `stop-cluster.sh` 脚本的设置需要重命名该文件。
- **完善 Flink 和 Docker 的集成**： `Dockerfiles` 文件样例和 `build.sh` Docker 镜像文件都从 Flink GitHub 仓库中移除了，这些示例社区不再提供，因此 `flink-contrib/docker-flink` 、 `flink-container/docker` 和 `flink-container/kubernetes` 模块都已删除了。目前你可以通过查看 Flink Docker integration 官方文档学会如何使用和自定义 Flink Docker 镜像，文档中包含了 docker run、docker compose、docker swarm 和 standalone Kubernetes。

内存管理

- **JobManager 使用新的内存模型**：可以参考 FLIP-116，介绍了 JobManager 新的内存模型，提供了新的配置选项来控制 JobManager 的进程内存消耗，这种改变会影响 Standalone、YARN、Mesos 和 Active Kubernetes。如果你尝试在不做任何调整的情况下重用以前的Flink 配置，则新的内存模型可能会导致 JVM 的计算内存参数不同，从而导致性能发生变化甚至失败，可以参考 Migrate Job Manager Memory Configuration 文档进行迁移变更。`jobmanager.heap.size` 和 `jobmanager.heap.mb` 配置参数已经过期了，如果这些过期的选项还继续使用的话，为了维持向后兼容性，它们将被解释为以下新选项之一：
  - jobmanager.memory.heap.size：JVM Heap，为了 Standalone 和 Mesos 部署
  - jobmanager.memory.process.size：进程总内存，为了容器部署（Kubernetes 和 YARN）

  下面两个选项已经删除了并且不再起作用了：
  - containerized.heap-cutoff-ratio

- containerized.heap-cutoff-min

JVM 参数，JobManager JVM 进程的 direct 和 metaspace 内存现在通过下面两个参数进行配置：

- jobmanager.memory.off-heap.size
- jobmanager.memory.jvm-metaspace.size

如果没有正确配置或存在相应的内存泄漏，这些新的限制可能会产生相应的 OutOfMemoryError 异常，可以参考 OutOfMemoryError 文档进行解决。

- **移除过期的** `mesos.resourcemanager.tasks.mem` **参数**

Table API/SQL

- **Blink planner 成为默认的 planner**
- **改变了 Table API 的包结构**：由于包 `org.apache.flink.table.api.scala/java` 中的各种问题，这些包下的所有类都已迁移。 此外，如 Flink 1.9 中所述，scala 表达式已移至 `org.apache.flink.table.api` 。

  如果你之前使用了下面的类：

  - `org.apache.flink.table.api.java.StreamTableEnvironment`
  - `org.apache.flink.table.api.scala.StreamTableEnvironment`
  - `org.apache.flink.table.api.java.BatchTableEnvironment`
  - `org.apache.flink.table.api.scala.BatchTableEnvironment`

  如果你不需要转换成 DataStream 或者从 DataStream 转换，那么你可以使用：

  - `org.apache.flink.table.api.TableEnvironment`

  如果你需要转换成 DataStream/DataSet，或者从 DataStream/DataSet 转换，那么你需要将依赖 imports 改成：

  - `org.apache.flink.table.api.bridge.java.StreamTableEnvironment`
  - `org.apache.flink.table.api.bridge.scala.StreamTableEnvironment`
  - `org.apache.flink.table.api.bridge.java.BatchTableEnvironment`
  - `org.apache.flink.table.api.bridge.scala.BatchTableEnvironment`

  对于 Scala 表达式，使用下面的 import：

  - `org.apache.flink.table.api._ instead of org.apache.flink.table.api.bridge.scala._`

  如果你使用 Scala 隐式转换成 DataStream/DataSet，或者从 DataStream/DataSet 转换，那么该导入

    - `org.apache.flink.table.api.bridge.scala._`
- **移除 StreamTableSink 接口中的 emitDataStream 方法**：该接口的 emitDataStream 方法将移除
- **移除 BatchTableSink 中的 emitDataSet 方法**：将该接口的 emitDataSet 方法重命名为 consumeDataSet 并且返回 DataSink
- **纠正** `TableEnvironment.execute()` ** 和** `StreamTableEnvironment.execute()` **的执行行为**：在早期的版本，`TableEnvironment.execute()` 和 `StreamExecutionEnvironment.execute()` 都可以触发 Table 程序和 DataStream 程序。从 Flink 1.11.0 开始，Table 程序只能由 `TableEnvironment.execute()` 触发。将 Table 程序转换为 DataStream 程序（通过 `toAppendStream()` 或 `toRetractStream()` 方法）后，只能由 `StreamExecutionEnvironment.execute()` 触发它。
- **纠正** `ExecutionEnvironment.execute()` **和** `BatchTableEnvironment.execute()` **的执行行为**：在早期的版本中，`BatchTableEnvironment.execute()` 和 `ExecutionEnvironment.execute()` 都可以触发 Table 和 DataSet 应用程序（针对老的 planner）。从 Flink 1.11.0 开始，批处理 Table 程序只能由 `BatchEnvironment.execute()` 触发。将 Table 程序转换为DataSet 程序（通过 toDataSet() 方法）后，只能由 `ExecutionEnvironment.execute()` 触发它。
- **在 Row 类型中添加了更改标志**：在 Row 类型中添加了一个更改标志 RowKind

配置

- **重命名** `log4j-yarn-session.properties` **和** `logback-yarn.xml` **配置文件**：日志配置文件 `log4j-yarn-session.properties` 和 `logback-yarn.xml` 被重命名为 `log4j-session.properties` 和 `logback-session.xml` 而且， `yarn-session.sh` 和 `kubernet -session.sh` 使用这些日志配置文件。

状态

- **删除已弃用的后台清理开关**： `StateTtlConfig#cleanupInBackground` 已经被删除，因为在 1.10 中该方法已被弃用，并且默认启用了后台 TTL。
- **删除禁用 TTL 压缩过滤器的的选项**：默认情况下，RocksDB 中的 TTL 压缩过滤器在 1.10 中是启用的，在 1.11+ 中总是启用的。因此，在 1.11 中删除了以下选项和方法：
    - state.backend.rocksdb.ttl.compaction.filter.enabled
    - StateTtlConfig#cleanupInRocksdbCompactFilter()
    - RocksDBStateBackend#isTtlCompactionFilterEnabled
    - RocksDBStateBackend#enableTtlCompactionFilter

- RocksDBStateBackend#disableTtlCompactionFilter
- (state_backend.py) is_ttl_compaction_filter_enabled
- (state_backend.py) enable_ttl_compaction_filter
- (state_backend.py) disable_ttl_compaction_filter

- **改变 `StateBackendFactory#createFromConfig` 的参数类型**：从 Flink 1.11 开始，`StateBackendFactory` 接口中的 `createFromConfig` 方法中的参数变为 ReadableConfig 而不是 Configuration。Configuration 类是 ReadableConfig 接口的实现类，因为它实现了 ReadableConfig 接口，所以自定义 StateBackend 也应该做相应的调整。
- **删除过期的 `OptionsFactory` 和 `ConfigurableOptionsFactory` 类**：过期的 OptionsFactory 和 ConfigurableOptionsFactory 类已被删除。请改用 RocksDBOptionsFactory 和 ConfigurableRocksDBOptionsFactory。如果任何类扩展了 DefaultConfigurableOptionsFactory，也请重新编译你的应用程序代码。
- **默认情况下启用** setTotalOrderSeek：从 Flink 1.11 开始，默认情况下，RocksDB 的 ReadOptions 将启用 setTotalOrderSeek 选项。这是为了防止用户忘记使用 optimizeForPointLookup。为了向后兼容，我们支持通过 RocksDBOptionsFactory 自定义 ReadOptions。如果观察到性能下降，请将 setTotalOrderSeek 设置为 false（根据我们的测试，这种情况不应该发生）。
- **增加 `state.backend.fs.memory-threshold` 的默认值**：`state.backend.fs.memory-threshold` 的默认值已从 1K 增加到20K，以防止在远程 FS 上为小状态创建太多小文件。对于那些 source 处配置很多并行度或者有状态的算子的作业可能会因此变更而出现 `JM OOM` 或 `RPC message exceeding maximum frame size` 的问题。如果遇到此类问题，请手动将配置设置回 1K。

PyFlink

- **对于不支持的数据类型将抛出异常**：可以使用一些参数（例如，精度）来配置数据类型。但是，在以前的版本中，用户提供的精度没有任何效果，会使用该精度的默认值。为了避免混淆，从 Flink 1.11 开始，如果不支持该数据类型，则将引发异常。 更改包括：
  - `TimeType` 精度只能为 `0`
  - `VarBinaryType` / `VarCharType` 的长度是 `0x7fffffff`
  - `DecimalType` 可选值是 `38` / `18`
  - `TimestampType` / `LocalZonedTimestampType` 的精度只能是 `3`
  - `DayTimeIntervalType` 的单位是 `SECOND` ， `fractionalPrecision` 精度只能为 `3`
  - `YearMonthIntervalType` 的单位是 `MONTH` ， `yearPrecision` 精度只能为 `2`
  - `CharType` / `BinaryType` / `ZonedTimestampType` 不支持

监控

- **将所有的 MetricReporters 转换为 plugins**：Flink 的所有 MetricReporters 都已经转换为 plugins，它们不再存放在 lib 目录下（这样做可能会导致依赖冲突），而应该放到 `/plugins/<some_directory>` 目录下。
- **改变 DataDog 的 metrics reporter Counter Metrics**：现在 DataDog metrics reporter 程序将 Counter 指标上报为报告时间间隔内的事件数，而不是总数，将 Counter 语义与 DataDog 文档保持一致。
- **切换 Log4j2 为默认的**：Flink 现在默认使用 Log4j2，希望恢复到 Log4j1 的用户可以在日志文档中找到操作说明
- **更改 JobManager API 的日志请求行为**：从 JobManager 服务端请求一个不可用的 log 或者 stdout 文件现在会返回 404 状态码，在之前的版本中，会返回 `file unavailable` 。
- **移除 lastCheckpointAlignmentBuffered metric**：现在 lastCheckpointAlignmentBuffered metric 已经被移除了，因为在发出 Checkpoint barrier 之后，上游的任务不会发送任何数据，直到下游侧完成对齐为止，WebUI 仍然会显示该值，但现在始终为 0。

Connectors

- **移除 Kafka 0.8/0.9 Connector**
- **移除 ElasticSearch 2.x Connector**
- **移除** KafkaPartitioner
- **改进的 fallback 文件系统，以只处理特定的文件系统**
- **将 `FileSystem#getKind` 方法设置过期的**

Runtime

- **流作业在 Checkpoint 同步部分失败时会立即失败**：无论配置什么参数，Checkpoint 同步部分中的失败（如算子抛出异常）都将立即使其任务（和作业）失败，从 Flink 1.5 版本开始，可以通过设置 `setTolerableCheckpointFailureNumber(...)` 或 `setFailTaskOnCheckpointError(...)` 参数来忽略此类的失败，现在这两个参数只影响异步的失败。
- **Checkpoint 超时不再被 `CheckpointConfig#setTolerableCheckpointFailureNumber` 忽略**：现在将 Checkpoint 超时视为正常的 Checkpoint 故障，并根据 `CheckpointConfig#setTolerableCheckpointFailureNumber` 配置的值进行检查。

各种接口变更

- **移除过期的** `StreamTask#getCheckpointLock()` ：在方法在 Flink 1.10 中已经设置过期了，目前不再提供该方法。用户可以使用 MailboxExecutor 来执行需要与任务线程安全的操作。
- **flink-streaming-java 模块不再依赖 flink-client 模块**：从 Flink 1.11.0 开始，flink-streaming-java 模块不再依赖 flink-client 模块，如果你项目依赖于 flink-client 模块，需要显示的添加其为依赖项。
- **AsyncWaitOperator 是可链接的**：默认情况下，将允许 AsyncWaitOperator 与所有算子链接在一起，但带有 SourceFunction 的任务除外。
- 更改了 `ShuffleEnvironment` 接口的 `createInputGates` 和 `createResultPartitionWriters` 方法的参数类型。
- `CompositeTypeSerializerSnapshot#isOuterSnapshotCompatible` 方法标示过期了。
- 移除了过期的 TimestampExtractor：可以使用 TimestampAssigner 和 WatermarkStrategies。
- 将 `ListCheckpointed` 标示为过期的：可以使用 CheckpointedFunction 作为代替
- 移除了过期的 state 连接方法：移除了 `RuntimeContext#getFoldingState()` 、`OperatorStateStore#getSerializableListState()` 和 `OperatorStateStore#getOperatorState()` 连接状态的方法，这意味着在 1.10 运行成功的代码在 1.11 上是运行不了的。

详情参考 https://ci.apache.org/projects/flink/flink-docs-master/release-notes/flink-1.11.html