第十二章 Flink

第十二章考点

- Flink主要特性
- 流处理特点
- Flink理想框架
- Flink优势

1. Flink 概述与核心特性

1.1 什么是 Flink

- **定义**：Apache Flink 是一个分布式、高性能、高可用的开源**流处理框架**。
- **核心理念**：同时支持**实时计算**和**批量计算**（批流一体）。
- **起源**：起源于 Stratosphere 项目，实现了 Google Dataflow 流计算模型。

1.2 核心特性

1. **批流一体化**：将批处理看作是流处理的一种特殊情况（有界流）。
2. **高性能**：兼具**高吞吐**和**低延迟**。
3. **精确一次 (Exactly-once)**：提供状态一致性保障，确保数据不丢不重。
4. **状态管理**：支持 TB 级别的状态数据，支持有状态计算。
5. **事件时间 (Event Time)**：支持基于事件发生时间的处理，解决乱序问题。
6. **独立内存管理**：不完全依赖 JVM GC，通过序列化将对象转为二进制存储，有效利用内存。

1.3 架构演变

1. **传统数据处理架构**：
   - 中心化数据库存储事务数据。
   - **缺点**：数据库不堪重负，难以扩展。
2. **大数据 Lambda 架构**：
   - **双层架构**：批处理层 (MapReduce/Spark) + 实时处理层 (Storm/Spark Streaming)。
   - **缺点**：多套框架难管理，代码需要维护两套。
3. **流处理架构 (Flink 推崇)**：
   - 一方面，由于流处理架构中不存在一个大型集中式数据库，因此，避免了传统 数据处理架构中存在的“数据库不堪重负”的问题。
   - 另一方面，在流处理架构中，批处理被看成是流处理的一个子集，因此，就可 以用面向流处理的框架进行批处理，这样就可以用一个流处理框架来统一处理流 计算和批量计算，避免了Lambda架构中存在的“多个框架难管理”的问题题。

1.4 主流框架对比 (重点)

框架	延迟	吞吐量	一致性/容错	处理模型
Storm	**极低**	低	At-least-once (较弱)	逐条处理
Spark Streaming	秒级/高	**高**	Exactly-once	**微批处理** (Micro-batch)
Flink	**极低**	**高**	**Exactly-once** (强)	**逐条流式处理**

- **结论**：Flink 是目前唯一同时满足高吞吐、低延迟、精确一次容错的理想框架。

1.5 Flink的优势

- 同时支持高吞吐、低延迟、高性能。
- 同时支持流处理和批处理
- 高度灵活的流式窗口。
- 支持有状态计算
- 具有良好的容错性
- 具有独立的内存管理：为了获得C一样的性能，Flink管理Java虚 拟机的内存。通过序列化与反序列化的方法将所有数据对象转换成二进 制在内存中进行存储，能够降低数据存储空间，有效利用内存空间。
- 支持迭代和增量迭代。 增量迭代：对某些迭代而言，并不是单次迭代产生的下一次工作集中的每个 元素都需要重新参与下一轮迭代，有时只需要重新计算部分数据同时选择性 地更新解集。增量迭代能使一些算法执行得更高效。

---

2. Flink 的三大应用场景

2.1 事件驱动型应用 (Event-driven)

- **定义**：从事件流读取，根据事件触发计算、状态更新或外部动作。
- **典型案例**：反欺诈、异常检测、规则报警。
- **Flink 优势**：
  - **Savepoint (保存点)**：一致性的状态镜像，支持应用升级或扩容而不丢失状态。
  - **CEP 库**：复杂事件处理支持。

2.2 数据分析应用 (Data Analytics)

- **定义**：从原始数据提取信息（如报表、仪表盘）。
- **优势**：
  - 相比批量分析，流式分析延迟更低（实时洞察）。
  - 架构更简单（利用 Flink 的故障恢复机制，无需复杂的调度流水线）。
- **支持**：ANSI SQL 接口，批流一致的 SQL 语义。

2.3 数据流水线应用 (Data Pipeline)

- **定义**：类似 ETL，但以连续流模式执行，而非周期性触发。
- **典型案例**：实时搜索索引构建、数据持续同步。
- **优势**：降低数据迁移延迟。
- **支持**：丰富的 Connectors (Kafka, JDBC, ES 等)。

---

3. Flink 技术栈与编程模型

3.1 体系架构 (Master-Slave)

- **JobManager (Master)**：负责资源分配、任务调度、Checkpoint 协调。
- **TaskManager (Slave)**：负责具体的任务执行 (Task)，拥有独立的 JVM。

3.2 技术栈分层

1. **物理部署层**：Local, Cluster (Standalone, YARN), Cloud。
2. **Runtime 核心层**：分布式数据流引擎。
3. **API 层**：
   - **DataStream API**：用于**流处理**。
   - **DataSet API**：用于**批处理**。
4. **库层 (Libraries)**：
   - **CEP**：复杂事件处理。
   - **Table API & SQL**：关系型查询。
   - **Gelly**：图计算。
   - **FlinkML**：机器学习。

3.3 编程抽象级别 (由高到低)

1. **SQL** (最高层，声明式)。
2. **Table API** (声明式 DSL)。
3. **DataStream / DataSet API** (核心 API，通用性强)。
4. **有状态数据流处理** (最底层，如 ProcessFunction，控制力最强)。

---

5. 编程实践 (WordCount 逻辑)

5.1 开发流程

1. **环境准备**：安装 Java8+，安装 Maven。
2. **编写代码** (Maven 项目)。

3. **打包：** `mvn package` 生成 JAR 包。
4. **提交任务：** `./bin/flink run ...`。

5.2 核心代码逻辑 (Java)

一个标准的 Flink 程序包含以下步骤：

1. **获取执行环境：**
```
ExecutionEnvironment env =
ExecutionEnvironment.getExecutionEnvironment();
```
2. **加载/创建初始数据：**
```
// 批处理读取文件
DataSet<String> text = env.readTextFile("path/to/input");
```
3. **指定转换操作 (Transformation)：**
```
AggregateOperator counts = text
    .flatMap(new WordCountTokenizer()) // 切分单词
    .groupBy(0)                        // 按单词分组
    .sum(1);                           // 统计频次
```
4. **指定计算结果输出 (Sink)：**
```
counts.print(); // 或 writeAsCsv
```
5. **触发程序执行** (流处理必须，批处理视情况)：
```
env.execute();
```

复习重点总结 (记忆口诀)

- **定位：**批流一体，低延迟高吞吐。
- **架构：**JobManager + TaskManager。
- **优势：**对比 Spark Streaming 是真正的实时（逐条），对比 Storm 有状态和强一致性。
- **状态：**Checkpoints (容错), Savepoints (运维)。
- **API：**SQL > Table > DataStream/DataSet。