

数据湖探索

产品公告

文档版本 01
发布日期 2025-02-27



版权所有 © 华为技术有限公司 2025。保留一切权利。

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

商标声明



HUAWEI和其他华为商标均为华为技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

安全声明

漏洞处理流程

华为公司对产品漏洞管理的规定以“漏洞处理流程”为准，该流程的详细内容请参见如下网址：

<https://www.huawei.com/cn/psirt/vul-response-process>

如企业客户须获取漏洞信息，请参见如下网址：

<https://securitybulletin.huawei.com/enterprise/cn/security-advisory>

目录

1 产品公告	1
1.1 DLI Spark 3.1.1 版本停止服务（EOS）公告	1
1.2 DLI “包年/包月”和“按需计费”计费模式队列和“队列 CU 时套餐包”下线（EOL）公告	3
1.3 DLI 经典型跨源连接下线（EOL）公告	5
1.4 DLI Flink 1.10、Flink 1.11 版本停止服务（EOS）公告	6
1.5 DLI Spark 2.3.2 版本停止服务（EOS）公告	7
1.6 DLI Flink1.7 版本停止服务（EOS）公告	8
2 版本支持公告	10
2.1 DLI 计算引擎版本生命周期	10
2.2 Flink 1.15 版本说明	11
2.3 Flink 1.12 版本说明	12
2.4 Spark 3.3.1 版本说明	13
2.5 Spark 3.1.1 版本说明	14
2.6 Spark 2.4.5 版本说明	15
2.7 Spark 2.4.x 与 Spark 3.3.x 版本差异对比	16
2.7.1 Spark 2.4.x 与 Spark 3.3.x 版本在 SQL 队列的差异对比	16
2.7.2 Spark 2.4.x 与 Spark 3.3.x 版本在通用队列的差异对比	33
2.7.3 DLI datasourceV1 表和 datasourceV2 表	42

1 产品公告

1.1 DLI Spark 3.1.1 版本停止服务（EOS）公告

公告说明

华为云在2024年12月31 00:00（北京时间）将Spark 3.1.1版本停止服务（EOS）。

影响

DLI Spark 3.1.1版本停止服务（EOS）后，不再提供该软件版本的任何技术服务支持。建议您在执行作业时选择新版本的Spark引擎，推荐使用DLI Spark 3.3.1版本。

正在使用DLI Spark 3.1.1版本的作业也请您尽快切换至新版本的Spark引擎，否则作业执行过程中出现的错误，不再提供该版本的任何技术服务支持。

如您有任何问题，可随时通过[工单](#)或者服务热线（4000-955-988或950808）与我们联系。

常见问题

- DLI Spark 3.1.1版本停止服务后，正在使用该版本的作业有哪些影响？**
在Spark 3.1.1 EOS时间后创建的队列，在执行作业时不支持选择已经EOS的计算引擎。
历史创建的队列仍然可以使用Spark 2.3.2执行作业，但作业执行过程中出现的错误，不再提供该版本的任何技术服务支持，请您尽快更换至新版本的计算引擎。
- DLI Spark 3.1.1版本停止服务后，可以使用哪个版本替换？**
推荐使用DLI Spark 3.3.1版本。
- DLI Spark 3.3.1版本有哪些优势？**

表 1-1 Spark 3.3.1 版本优势

特性	说明
Native性能加速	Spark查询语句性能提升。

特性	说明
元数据访问性能提升	提升Spark在处理大数据时的元数据访问性能，提高数据处理流程效率。
提升OBS committer小文件写性能	提升对象存储服务（OBS）在处理小文件写入时的性能，提高数据传输效率。
动态Executor shuffle数据优化	提升资源扩缩容的稳定性，当shuffle文件不需要时清理Executor。
支持配置小文件合并	使用SQL过程中，生成的小文件过多时，会导致作业执行时间过长，且查询对应表时耗时增大，建议对小文件进行合并。 参考 如何合并小文件 完成合并小文件。
支持修改非分区表或分区表的列注释	修改非分区表或分区表的列注释。
支持统计SQL作业的CPU消耗	支持在控制台查看“CPU累计使用量”。
支持容器集群Spark日志跳转查看	需要在容器查看日志。
支持动态加载UDF（公测）	无需重启队列UDF即可生效。
Spark UI支持火焰图	Spark UI支持绘制火焰图。
优化SQL作业NOT IN语句查询性能	NOT IN语句查询性能提升。
优化Multi-INSERT语句查询性能	Multi-INSERT语句查询性能提升。

- **切换至新版本对DLI资源价格是否有影响？**
DLI按作业运行所需的计算资源和存储资源计费，与计算引擎版本无关。
- **如何升级到DLI Spark 3.1.1版本？**
 - a. 在DLI管理控制台，购买弹性资源池和队列资源，用于提供执行作业所需的计算资源。
 - b. 单击“作业管理 > Spark作业”，新建Spark作业。
 - c. 在作业管理页面，单击作业操作列的“编辑”，进入作业运行参数编辑页面。
 - d. 配置Spark版本，选择最新的Spark引擎版本，推荐使用Spark 3.3.1。

1.2 DLI “包年/包月”和“按需计费”计费模式队列和“队列CU时套餐包”下线（EOL）公告

公告说明

为了更好的实现资源共享，提高计算资源利用率，DLI将“包年/包月”和“按需计费”计费模式队列升级为“弹性资源池队列”。即使用DLI计算资源需先购买弹性资源池，并在弹性资源池中创建队列。

- DLI “包年/包月”和“按需计费”计费模式队列和“队列CU时套餐包”在2024年3月31日 00:00（北京时间）停止销售（EOM）。
- DLI “包年/包月”和“按需计费”计费模式队列和“队列CU时套餐包”计划于2025年6月30日 00:00（北京时间）下线（EOL）。

影响

- EOM后，不再支持新购队列资源，包括不支持新购“包年/包月”和“按需计费”计费模式队列和“队列CU时套餐包”。

为了满足部分业务的使用需求，在2024年6月30日 00:00:00（北京时间）前，您可以最大续订1年队列或者变更队列。

2024年6月30日 00:00:00（北京时间）之后将无法续订、变更队列。

- EOL后，队列资源将无法继续使用，请务必在下线前更换使用“弹性资源池”或“default队列”。推荐您购买弹性资源池，并在弹性资源池中创建队列，体验更丰富的DLI产品能力。

如您有任何问题，可随时通过[工单](#)或者服务热线（4000-955-988或950808）与我们联系。

DLI “包年/包月”和“按需计费”计费模式队列生命周期

表 1-2 DLI “包年/包月”和“按需计费”计费模式队列生命周期

功能	状态	EOM时间	EOL时间
DLI “包年/包月”和“按需计费”计费模式队列功能	EOM	2024年3月31日	2025年6月30日

📖 说明

- EOM: End of Marketing, 停止该功能销售。
- EOL: End of life, 停止所有销售、服务活动。

常见问题

- DLI “包年/包月”和“按需计费”计费模式队列停止销售（EOM）后，正在使用该功能的作业有哪些影响？

EOM后停止队列资源的新购。

- 短期方案：为了满足部分业务的使用需求，在2024年6月30日 00:00:00（北京时间）前，您可以最大续订1年队列或者变更队列，在2024年6月30日 00:00:00（北京时间）后将无法续订、变更队列，
- 长期方案：更换使用“弹性资源池”或“default队列”，推荐您购买弹性资源池，并在弹性资源池中创建队列。

- **DLI“包年/包月”和“按需计费”计费模式队列下线（EOL）后，正在使用该功能的作业有哪些影响？**

EOL后，将无法使用“包年/包月”和“按需计费”计费模式队列执行作业。

请您务必在EOL前完成将“包年/包月”和“按需计费”计费模式队列上执行的作业迁移至弹性资源池队列。即用户重新购买弹性资源池，使用新购的弹性资源池队列执行作业。

- **DLI“包年/包月”和“按需计费”计费模式队列在EOM和EOL后，可以使用哪个功能替换？**

如果您正在使用“包年/包月”和“按需计费”计费模式队列运行作业，请尽快更换使用“弹性资源池”或“default队列”。

- 使用“按需专属”计费模式的队列可以直接迁移至新购的弹性资源池中。
- 使用“包年/包月”和“按需计费-非专属”的队列，请退订队列资源后，重新购买弹性资源池，使用新购的弹性资源池执行作业。

- **“弹性资源池队列”对比“包年/包月”和“按需计费”计费模式队列有哪些优势？**

- “包年/包月”和“按需计费”队列：固定资源规格，当作业任务对资源的请求量发生变化时，会导致队列资源浪费或者资源不足的问题。
- 弹性资源池队列：动态扩缩容，提高资源利用效率。

- **弹性资源池收费吗？弹性资源池中的队列收费吗？**

弹性资源池支持按需、包周期、套餐包的计费模式。您可以参考DLI计费说明了解更多弹性资源池计费信息。

添加到弹性资源池中的队列不再单独计费，以弹性资源池为计费项计费：

- 按需计费模式：按弹性资源池的实际CUs计费。
- 包年/包月计费模式：按弹性资源池的实际CUs计费，其中规格部分包年/包月，超出规格的部分按需计费。
- 弹性资源池CU时套餐包：按购买的套餐包周期价格扣费，其中套餐包内的规格包周期计费，超出套餐包规格的部分按需计费。

更多弹性资源池计费信息请参考[弹性资源池计费](#)。

- **怎样创建“弹性资源池队列”？**

a. **购买弹性资源池。**

- i. 在DLI管理控制台，单击“资源管理 > 弹性资源池”。
- ii. 在弹性资源池管理界面，单击界面右上角的“购买弹性资源池”。
按需配置弹性资源池的相关参数，完成弹性资源池的购买。

b. **在弹性资源池中创建队列。**

创建完弹性资源池后，弹性资源池可以添加一个或多个队列用于后续作业的运行。

- i. 在弹性资源池管理页面。

- ii. 选择要操作的弹性资源池，在“操作”列，单击“添加队列”。
 - iii. 在“添加队列”界面，完成队列的基础信息配置。单击“确定”完成添加队列配置。
- c. **创建作业。**
队列创建完成后，就可以使用弹性资源池中创建的队列进行作业配置。

公告发布日期：2023年11月21日

1.3 DLI 经典型跨源连接下线（EOL）公告

公告说明

华为云在2024年06月30 00:00（北京时间）将DLI经典型跨源连接的相关功能下线（EOL）。

影响

DLI经典型跨源连接的相关功能下线（EOL）后，不再提供该功能任何技术服务与支持。

如果您正在使用DLI经典型跨源连接的功能，请尽快更换使用[增强型跨源连接](#)，否则使用经典型跨源连接操作过程中出现的错误，不再提供该功能的任何技术服务支持。

如您有任何问题，可随时通过[工单](#)或者服务热线（4000-955-988或950808）与我们联系。

DLI 经典型跨源连接生命周期

表 1-3 DLI 经典型跨源连接生命周期

功能	状态	EOM时间	EOL时间
DLI经典型跨源连接	计划下线，不推荐使用	2023年12月30日	2024年6月30日

说明

- EOM：End of Marketing，停止该功能销售。
- EOL：End of life，停止所有销售、服务活动。

常见问题

- **DLI经典型跨源连接EOM和EOL后，正在使用该功能的作业有哪些影响？**
如果您正在使用DLI经典型跨源连接的功能，请尽快更换使用[增强型跨源连接](#)。否则使用经典型跨源连接操作过程中出现的错误，不再提供该功能的任何技术服务支持。
- **DLI经典型跨源连接停止服务后，可以使用哪个功能替换？**
推荐使用[增强型跨源连接](#)。

DLI提供的增强型跨源连接功能，通过建立对等连接，实现DLI与数据源的VPC网络互通，为DLI的跨源分析提供了支持。

目前DLI支持跨源访问的数据源请参考[跨源分析开发方式](#)。

- **增强型跨源连接收费吗？**

增强型跨源连接不计费。

- **怎样创建增强型跨源连接？**

a. 在DLI管理控制台，单击“跨源管理 > 增强型跨源”。

b. 选择“增强型跨源”，单击“创建”。

配置增强型跨源连接的信息请参考创建[增强型跨源连接](#)。

创建完成后，增强型跨源连接的状态显示“已激活”，代表该连接创建成功。您还可以在队列管理页面测试数据源与DLI队列的连通性。

公告发布日期：2023年11月10日

1.4 DLI Flink 1.10、Flink 1.11 版本停止服务（EOS）公告

公告说明

华为云在2023年12月31 00:00（北京时间）将DLI Flink 1.10、Flink1.11版本停止服务（EOS）。

影响

DLI Flink 1.10、Flink1.11版本停止服务（EOS）后，不再提供该软件版本的任何技术服务支持。建议您在执行作业时选择新版本的Flink引擎，推荐使用DLI Flink 1.15版本。

正在使用Flink 1.10、Flink1.11版本的作业也请您尽快切换至新版本的Flink引擎，否则作业执行过程中出现的错误，不再提供该版本的任何技术服务支持。

如您有任何问题，可随时通过[工单](#)或者服务热线（4000-955-988或950808）与我们联系。

常见问题

- **Flink 1.10、Flink1.11版本停止服务后，正在使用该版本的作业有哪些影响？**

在Flink 1.10、Flink1.11EOS时间后创建的对队列，在执行作业时不支持选择已经EOS的计算引擎。

历史创建的队列仍然可以使用Flink 1.10、Flink1.11执行作业，但作业执行过程中出现的错误，不再提供该版本的任何技术服务支持，请您尽快更换至新版本的计算引擎。

- **Flink 1.10、Flink1.11版本停止服务后，可以使用哪个版本替换？**

推荐使用DLI Flink 1.15版本。

- **Flink 1.15版本有哪些优势？**

- Flink 1.15版本在语法设计上实现了更高的兼容性，与主流开源技术标准保持一致。

- Flink 1.15版本新增读写Hive、Hudi等Connector。

更多Flink 1.15版本的优势请参考[Flink 1.15升级指导](#)。

- **切换至新版本计算引擎对DLI资源价格是否有影响？**
DLI按作业运行所需的计算资源和存储资源计费，与计算引擎版本无关。
- **如何升级到Flink 1.15版本？**
 - a. 在DLI管理控制台，购买弹性资源池和队列资源，用于提供执行作业所需的计算资源。
 - b. 在DLI管理控制台，单击“作业管理 > Flink作业”。
 - c. 在作业管理页面，单击作业操作列的“编辑”，进入作业运行参数编辑页面。
 - d. 配置Flink版本，选择最新的Flink版本，推荐使用Flink 1.15。
Flink 1.15版本语法请参考[Flink OpenSource SQL1.15版本使用说明](#)。

公告发布日期：2023年7月6日

1.5 DLI Spark 2.3.2 版本停止服务（EOS）公告

公告说明

华为云在2023年12月31 00:00（北京时间）将Spark 2.3.2版本停止服务（EOS）。

影响

DLI Spark 2.3.2版本停止服务（EOS）后，不再提供该软件版本的任何技术服务支持。建议您在执行作业时选择新版本的Spark引擎，推荐使用DLI Spark 3.3.1版本。

正在使用DLI Spark 2.3.2版本的作业也请您尽快切换至新版本的Spark引擎，否则作业执行过程中出现的错误，不再提供该版本的任何技术服务支持。

如您有任何问题，可随时通过[工单](#)或者服务热线（4000-955-988或950808）与我们联系。

常见问题

- **DLI Spark 2.3.2版本停止服务后，正在使用该版本的作业有哪些影响？**
在Spark 2.3.2 EOS时间后创建的队列，在执行作业时不支持选择已经EOS的计算引擎。
历史创建的队列仍然可以使用Spark 2.3.2执行作业，但作业执行过程中出现的错误，不再提供该版本的任何技术服务支持，请您尽快更换至新版本的计算引擎。
- **DLI Spark 2.3.2版本停止服务后，可以使用哪个版本替换？**
推荐使用DLI Spark 3.3.1版本。
- **DLI Spark 3.3.1版本有哪些优势？**

表 1-4 Spark 3.3.1 版本优势

特性	说明
Native性能加速	Spark查询语句性能提升。
元数据访问性能提升	提升Spark在处理大数据时的元数据访问性能，提高数据处理流程效率。

特性	说明
提升OBS committer小文件写性能	提升对象存储服务（OBS）在处理小文件写入时的性能，提高数据传输效率。
动态Executor shuffle数据优化	提升资源扩缩容的稳定性，当shuffle文件不需要时清理Executor。
支持配置小文件合并	使用SQL过程中，生成的小文件过多时，会导致作业执行时间过长，且查询对应表时耗时增大，建议对小文件进行合并。 参考 如何合并小文件 完成合并小文件。
支持修改非分区表或分区表的列注释	修改非分区表或分区表的列注释。
支持统计SQL作业的CPU消耗	支持在控制台查看“CPU累计使用量”。
支持容器集群Spark日志跳转查看	需要在容器查看日志。
支持动态加载UDF（公测）	无需重启队列UDF即可生效。
Spark UI支持火焰图	Spark UI支持绘制火焰图。
优化SQL作业NOT IN语句查询性能	NOT IN语句查询性能提升。
优化Multi-INSERT语句查询性能	Multi-INSERT语句查询性能提升。

- **切换至新版本对DLI资源价格是否有影响？**
DLI按作业运行所需的计算资源和存储资源计费，与计算引擎版本无关。
- **如何升级到DLI Spark 2.4.5版本？**
 - a. 在DLI管理控制台，购买弹性资源池和队列资源，用于提供执行作业所需的计算资源。
 - b. 单击“作业管理 > Spark作业”，新建Spark作业。
 - c. 在作业管理页面，单击作业操作列的“编辑”，进入作业运行参数编辑页面。
 - d. 配置Spark版本，选择最新的Spark引擎版本，推荐使用Spark 3.3.1。

公告发布日期：2023年7月6日

1.6 DLI Flink1.7 版本停止服务（EOS）公告

公告说明

华为云在2022年12月31 00:00（北京时间）将DLI Flink 1.7版本停止服务（EOS）。

影响

DLI Flink 1.7版本停止服务（EOS）后，不再提供该软件版本相关的任何技术服务支持。建议您在执行作业时选择新版本的Flink引擎，推荐使用DLI Flink 1.15版本。

正在使用Flink 1.7版本的作业也请您尽快切换至新版本的Flink引擎，否则作业执行过程中出现的错误，不再提供该版本的任何技术服务支持。

如您有任何问题，可随时通过[工单](#)或者服务热线（4000-955-988或950808）与我们联系。

常见问题

- **Flink 1.7版本中有哪些重要功能不再演进？**
 - Flink Edge SQL功能随Flink1.7 EOS，后续版本的Flink均不再支持边缘场景的作业处理。
 - 敏感变量功能随Flink1.7 EOS，后续版本不再支持敏感变量。
- **Flink 1.7版本停止服务后，正在使用该版本的作业有哪些影响？**

在Flink 1.7 EOS后创建的队列，在执行作业时不支持选择已经EOS的计算引擎。历史创建的队列使用Flink 1.7执行作业过程中出现的错误，不再提供该版本的任何技术服务支持，请您尽快更换至新版本的计算引擎。
- **Flink 1.7版本停止服务后，可以使用哪个版本替换？**

推荐使用DLI Flink 1.15版本。
- **Flink 1.12版本有哪些优势？**

Flink 1.12新增支持DataGen源表、DWS源表、JDBC源表、MySQL CDC源表、Postgres CDC源表、Redis源表、Upsert Kafka源表、Hbase源表
更多升级Flink 1.12版本的优势请参考[Flink 1.12升级指导](#)。
- **切换至新版本计算引擎对DLI资源价格是否有影响？**

DLI按作业运行所需的计算资源和存储资源计费，与计算引擎版本无关。
- **如何升级到Flink 1.12版本？**
 - a. 在DLI管理控制台，购买弹性资源池和队列资源，用于提供执行作业所需的计算资源。
 - b. 在DLI管理控制台，单击“作业管理 > Flink作业”。
 - c. 在作业管理页面，单击作业操作列的“编辑”，进入作业运行参数编辑页面。
 - d. 配置Flink版本，选择最新的Flink版本，推荐使用Flink 1.15。

公告发布日期：2023年7月6日

2 版本支持公告

2.1 DLI 计算引擎版本生命周期

版本号说明

DLI计算引擎版本号：格式为**计算引擎名称 x.y.z**，其中计算引擎分为Flink和Spark，版本号具体含义如**图2-1**所示。

图 2-1 DLI 计算引擎版本号



版本支持情况

- Flink计算引擎推荐版本：Flink 1.15。
- Spark计算引擎推荐版本：Spark 3.3.1。

📖 说明

不建议长期混用不同版本的Spark/Flink引擎。

- 长期混用不同版本的Spark/Flink引擎会导致代码在新旧版本之间不兼容，影响作业的执行效率。
- 当作业依赖于特定版本的库或组件，长期混用不同版本的Spark/Flink引擎可能会导致作业因依赖冲突而执行失败。

计算引擎版本生命周期

表2-1给出了DLI计算引擎版本生命周期，帮助您规划自己的版本更新节奏。

表 2-1 DLI 计算引擎版本生命周期

计算引擎类型	版本名称	状态	EOM时间	EOS时间
Flink	DLI Flink 1.15	已发布	2025年6月30日	2026年6月30日
	DLI Flink 1.12	EOS	2023年12月31日	2024年12月31日
	DLI Flink 1.11	EOS	2022年6月30日	2023年12月31日
	DLI Flink 1.10	EOS	2022年6月30日	2023年12月31日
	DLI Flink 1.7	EOS	2021年12月31日	2022年12月31日
Spark	DLI Spark 3.3.1	已发布	2025年6月30日	2026年6月30日
	DLI Spark 3.1.1	EOS	2023年12月31日	2024年12月31日
	DLI Spark 2.4.5	EOS	2023年12月31日	2024年12月31日
	DLI Spark 2.3.2	EOS	2022年6月30日	2023年12月31日

📖 说明

- EOM: End of Marketing, 停止该版本的销售。所有新购资源不再支持选择EOM的版本的引擎。
- EOS: End of Service & support, 停止该版本的服务, 建议您在执行作业时选择最新版本的引擎。在该日期之后, 不再提供该软件版本的任何技术服务支持。

2.2 Flink 1.15 版本说明

数据湖探索 (DLI) 遵循开源Flink计算引擎的发布一致性。本文介绍Flink 1.15版本所做的变更说明。

更多Flink 1.15版本说明请参考[Release Notes - Flink Jar 1.15](#)、[Flink OpenSource SQL1.15版本使用说明](#)。

Flink 1.15 版本发布时间

版本名称	发布时间	状态	EOM时间	EOS时间
DLI Flink 1.15	2023年6月	已发布	2025年6月30日	2026年6月30日

更多版本支持信息请参考[DLI计算引擎版本生命周期](#)。

Flink 1.15 版本说明

- Flink 1.15版本在语法设计上实现了更高的兼容性，与主流开源技术标准保持一致。
- Flink 1.15版本新增读写Hive、Hudi等Connector。
- Flink 1.15版本数据同步迁移场景，优先推荐使用DataArts的数据集成。
- Flink 1.15版本支持集成DEW-CSMS凭证管理，提供隐私保护方案。
- Flink 1.15版本支持Flink Jar作业最小化提交。

📖 说明

Flink Jar作业最小化提交是指Flink仅提交作业必须的依赖项，而不是整个Flink环境。通过将非Connector的Flink依赖项（以flink-开头）和第三方库（如Hadoop、Hive、Hudi、Mysql-cdc）的作用域设置为provided，可以确保这些依赖项不会被包含在Jar作业中，从而实现最小化提交，避免依赖包与flink内核中依赖包冲突：

- 仅Flink 1.15版本支持Flink Jar作业最小化提交，通过在运行优化参数中配置flink.dli.job.jar.minimize-submission.enabled=true可以开启最小化提交。
- Flink相关依赖作用域请使用provided，即在依赖中添加<scope>provided</scope>。主要包含org.apache.flink组下以flink-开头的非Connector依赖。
- Hadoop、Hive、Hudi、Mysql-cdc相关依赖，作用域请使用provided，即在依赖中添加<scope>provided</scope>。
- Flink源代码中只有明确标注了@Public或者@PublicEvolving的才是公开供用户调用的方法，DLI只对这些方法的兼容性做出产品保证。

2.3 Flink 1.12 版本说明

数据湖探索（DLI）遵循开源Flink计算引擎的发布一致性。本文介绍Flink 1.12版本所做的变更说明。

更多Flink 1.12版本说明请参考[Release Notes - Flink 1.12](#)。

Flink 1.12 版本发布时间

版本名称	发布时间	状态	EOM时间	EOS时间
DLI Flink 1.12	2021年12月	EOS	2023年12月31日	2024年12月31日

更多版本支持信息请参考[DLI计算引擎版本生命周期](#)。

Flink 1.12 版本说明

- Flink 1.12新增支持DataGen源表、DWS源表、JDBC源表、MySQL CDC源表、Postgres CDC源表、Redis源表、Upsert Kafka源表、Hbase源表。
- Flink 1.12新增支持小文件合并功能。
- Flink 1.12新增支持Redis维表、RDS维表。

2.4 Spark 3.3.1 版本说明

数据湖探索（DLI）遵循开源Spark计算引擎的发布一致性。本文介绍Spark 3.3.1版本所做的变更说明。

更多Spark 3.3.1版本说明请参考[Spark Release Notes](#)。

Spark 3.3.1 版本发布时间

版本名称	发布时间	状态	EOM时间	EOS时间
DLI Spark 3.3.1	2023年6月	已发布	2025年6月30日	2026年6月30日

更多版本支持信息请参考[DLI计算引擎版本生命周期](#)。

Spark 3.3.1 版本说明

[表2-2](#)列举了Spark 3.3.1 版本主要的功能特性。

更多版本新特性及性能优化请参考[Release Notes - Spark 3.3.1](#)。

表 2-2 Spark 3.3.1 版本优势

特性	说明
Native性能加速	Spark查询语句性能提升。
元数据访问性能提升	提升Spark在处理大数据时的元数据访问性能，提高数据处理流程效率。
提升OBS committer小文件写性能	提升对象存储服务（OBS）在处理小文件写入时的性能，提高数据传输效率。
动态Executor shuffle数据优化	提升资源扩缩容的稳定性，当shuffle文件不需要时清理Executor。
支持配置小文件合并	使用SQL过程中，生成的小文件过多时，会导致作业执行时间过长，且查询对应表时耗时增大，建议对小文件进行合并。 参考 如何合并小文件 完成合并小文件。
支持修改非分区表或分区表的列注释	修改非分区表或分区表的列注释。
支持统计SQL作业的CPU消耗	支持在控制台查看“CPU累计使用量”。
支持容器集群Spark日志跳转查看	需要在容器查看日志。

特性	说明
支持动态加载UDF（公测）	无需重启队列UDF即可生效。
Spark UI支持火焰图	Spark UI支持绘制火焰图。
优化SQL作业NOT IN语句查询性能	NOT IN语句查询性能提升。
优化Multi-INSERT语句查询性能	Multi-INSERT语句查询性能提升。

2.5 Spark 3.1.1 版本说明

数据湖探索（DLI）遵循开源Spark计算引擎的发布一致性。本文介绍Spark 3.1.1版本所做的变更说明。

更多Spark 3.1.1版本说明请参考[Spark Release Notes](#)。

Spark 3.1.1 版本发布时间

版本名称	发布时间	状态	EOM时间	EOS时间
DLI Spark 3.1.1	2021年12月	EOS	2023年12月31日	2024年12月31日

更多版本支持信息请参考[DLI计算引擎版本生命周期](#)。

Spark 3.1.1 版本说明

下表列举了Spark 3.1.1 版本主要的功能特性。

更多版本新特性请参考[Release Notes - Spark 3.1.1](#)。

- 【SPARK-33050】：Apache ORC升级到1.5.12。
- 【SPARK-33092】：增强子表达式消减。
- 【SPARK-33480】：支持char/varchar数据类型。
- 【SPARK-32302】：部分谓词下推优化。
- 【SPARK-30648】：支持JSON datasource表谓词下推。
- 【SPARK-32346】：支持avro datasource表谓词下推。
- 【SPARK-32461】：shuffle hash join优化。
- 【SPARK-32272】：添加SQL标准命令SET TIME_ZONE。
- 【SPARK-21492】：修复排序合并加入中的内存泄漏。
- 【SPARK-27812】：K8S客户端版本提升到4.6.1。

 说明

DLI从Spark 3.x版本开始不支持内置地理空间查询函数。

2.6 Spark 2.4.5 版本说明

数据湖探索（DLI）遵循开源Spark计算引擎的发布一致性。本文介绍Spark 2.4.5版本所做的变更说明。

更多Spark 2.4.5版本说明请参考[Spark Release Notes](#)。

Spark 2.4.5 版本发布时间

版本名称	发布时间	状态	EOM时间	EOS时间
DLI Spark 2.4.5	2021年12月	EOS	2023年12月31日	2024年12月31日

更多版本支持信息请参考[DLI计算引擎版本生命周期](#)。

Spark 2.4.5 版本说明

[表2-3](#)列举了Spark 2.4.5 版本主要的功能特性。

更多版本新特性请参考[Release Notes - Spark 2.4.5](#)。

表 2-3 Spark 2.4.5 版本优势

特性	说明
支持配置小文件合并	使用SQL过程中，生成的小文件过多时，会导致作业执行时间过长，且查询对应表时耗时增大，建议对小文件进行合并。 参考 如何合并小文件 完成合并小文件。
支持修改非分区表或分区表的列注释	修改非分区表或分区表的列注释。
支持统计SQL作业的CPU消耗	支持在控制台查看“CPU累计使用量”。
支持容器集群Spark日志跳转查看	需要在容器查看日志。
支持动态加载UDF（公测）	无需重启队列UDF即可生效。
Spark UI支持火焰图	Spark UI支持绘制火焰图。
优化SQL作业NOT IN语句查询性能	NOT IN语句查询性能提升。

特性	说明
优化Multi-INSERT语句查询性能	Multi-INSERT语句查询性能提升。

2.7 Spark 2.4.x 与 Spark 3.3.x 版本差异对比

2.7.1 Spark 2.4.x 与 Spark 3.3.x 版本在 SQL 队列的差异对比

DLI整理了Spark 2.4.x与Spark 3.3.x版本在SQL队列的差异，便于您了解Spark版本升级后SQL队列上运行的作业在适配新版本引擎时的影响。

histogram_numeric 函数的返回值的类型不同

- **说明:**
Spark SQL中的histogram_numeric函数返回一个结构体数组 (x,y)，不同版本的引擎x的类型不同。
 - **Spark2.4.x:** Spark 3.2或更早版本中，x为double类型。
 - **Spark3.3.x:** x类型等于函数输入值的类型。
- **升级引擎版本后是否对作业有影响:**
有影响，涉及相关用法需要适配。
- **示例代码:**
准备数据:

```
create table test_histogram_numeric(val int);
INSERT INTO test_histogram_numeric VALUES(1),(5),(8),(6),(7),(9),(8),(9);
```

执行sql:

```
select histogram_numeric(val,3) from test_histogram_numeric;
```

 - Spark 2.4.5
[{"x":1.0,"y":1.0},{"x":5.5,"y":2.0},{"x":8.200000000000001,"y":5.0}]
 - Spark 3.3.1
[{"x":1,"y":1.0},{"x":5,"y":2.0},{"x":8,"y":5.0}]

Spark3.3.x 不再支持使用“0\$”指定第一个参数

- **说明:**
format_string(strfmt, obj, ...) 和 printf(strfmt, obj, ...) 中的 strfmt 将不再支持使用“0\$”指定第一个参数，第一个参数应始终由“1\$”引用当使用参数索引来指示参数在参数列表中的位置。
 - **Spark2.4.x:** %0和%1均可表示第一个参数。
 - **Spark3.3.x:** 不再支持%0。
- **升级引擎版本后是否对作业有影响:**
有影响，请作业中如涉及使用%0需修改以适配Spark3.3.x。
- **示例代码1:**
执行sql:

```
SELECT format_string('Hello, %0$s! I\'m %1$s!', 'Alice', 'Lilei');
```

- Spark 2.4.5
Hello, Alice! I'm Alice!
- Spark 3.3.1
DLI.0005: The value of parameter(s) 'strfmt' in `format_string` is invalid: expects %1\$, %2\$ and so on, but got %0\$.
- **示例代码2:**
执行sql:
SELECT format_string('Hello, %1\$s! I\'m %2\$s!', 'Alice', 'Lilei');
- Spark 2.4.5
Hello, Alice! I'm Lilei!
- Spark 3.3.1
Hello, Alice! I'm Lilei!

Spark3.3.x 版本中空字符串无引号。

- **说明:**
默认情况下，空值在CSV数据源中，2.4.5版本空字符串为""，升级到spark3.3.1后空字符串无引号。
 - **Spark2.4.x:** 空值在CSV数据源中为""。
 - **Spark3.3.x:** 空值在CSV数据源中无引号。
如需在Spark3.3.x版本中恢复Spark2.4.x的格式，可以通过设置 spark.sql.legacy.nullValueWrittenAsQuotedEmptyStringCsv为 true来实现。
- **升级引擎版本后是否对作业有影响:**
有影响，导出orc文件中null值存储形式不同。
- **示例代码:**
准备数据:
create table test_null(id int,name string) stored as parquet;
insert into test_null values(1,null);
导出csv查看文件内容
 - Spark 2.4.5
1,""
 - Spark 3.3.1
1,

describe function 返回结果不同

- **说明:**
如果function不存在，describe function会执行失败。
 - **Spark2.4.x:** DESCRIBE函数仍然可以运行并打印“Function:func_name not found”
 - **Spark3.3.x:** 函数不存在的提示信息变更为失败。
- **升级引擎版本后是否对作业有影响:**
有影响，describe function 相关API的返回信息不同。
- **示例代码:**
执行sql:
describe function dli_no(dli_no不存在)
 - Spark 2.4.5

执行成功，function_desc内容：

```
Function:func_name not found
```

- Spark 3.3.1

执行失败，DLI.0005:

```
Undefined function: dli_no……
```

返回信息明确告知不支持指定表外部属性

- 说明：

表外部属性`external`变为保留。如果指定外部属性，某些命令将执行失败。

- **Spark2.4.x:** 通过`CREATE TABLE ... TBLPROPERTIES`和`ALTER TABLE ... SET TBLPROPERTIES`指定external属性，命令执行成功，但实际上external属性被静默忽略，表依然是managed table。

- **Spark3.3.x:**

通过`CREATE TABLE ... TBLPROPERTIES`和`ALTER TABLE ... SET TBLPROPERTIES`指定external属性，命令将会失败。

如需在Spark3.3.x版本中恢复Spark2.4.x的使用方式，可以通过设置spark.sql.legacy.notReserveProperties为true来实现。

- 升级引擎版本后是否对作业有影响：

有影响，涉及相关用法需要适配。

- 示例代码：

执行sql:

```
CREATE TABLE test_external(id INT,name STRING) TBLPROPERTIES('external'=true);
```

- Spark 2.4.5

执行成功。

- Spark 3.3.1

```
DLI.0005: The feature is not supported: external is a reserved table property, please use CREATE EXTERNAL TABLE.
```

新增支持解析“+Infinity”、“+INF”和“-INF”类型字符串的值

- 说明：

- **Spark2.4.x:** 当从定义为FloatType 或 DoubleType的JSON属性读取值时，Spark2.4.x仅支持解析“Infinity”和“-Infinity”。

- **Spark3.3.x:** 当从定义为FloatType 或 DoubleType的JSON属性读取值时，Spark3.3.x除了支持解析“Infinity”和“-Infinity”之外，还支持解析字符串“+Infinity”、“+INF”和“-INF”。

- 升级引擎版本后是否对作业有影响：

功能增强，无影响

默认配置 spark.sql.adaptive.enabled=true

- 说明：

- **Spark2.4.x:** 在Spark 2.4.x版本中，默认情况下spark.sql.adaptive.enabled配置项的值是false，即自适应查询执行（Adaptive Query Execution，简称AQE）特性是关闭的。

- **Spark3.3.x:** 从Spark3.3.x-320版本起开始默认开启AQE特性，即spark.sql.adaptive.enabled配置项的值是true。

- **升级引擎版本后是否对作业有影响：**
DLI功能增强，spark.sql.adaptive.enabled的默认参数值发生变化。

SHOW TABLES 输出的 schema 的变化

- **说明：**
SHOW TABLES的输出schema从database: string变成了namespace: string。
 - **Spark2.4.x:** SHOW TABLES的输出schema是database: string。
 - **Spark3.3.x:**
SHOW TABLES的输出schema从database: string变成了namespace: string。
其中对于内置catalog，namespace字段被命名为database；对于v2 catalog没有isTemporary字段。
如果你希望在Spark 3.3.x版本中恢复到Spark 2.4.x版本的样式，可以通过将spark.sql.legacy.keepCommandOutputSchema设置为true来实现。
- **升级引擎版本后是否对作业有影响：**
有影响，请排查作业中与SHOW TABLES有关的使用方法，并按上述说明适配新版本的使用要求。
- **示例代码：**
执行sql:

```
show tables;
```

- **Spark 2.4.5**

database	tableName	isTemporary
db1	table1	false
- **Spark 3.3.1**

namespace	tableName	isTemporary
db1	table1	false

SHOW TABLE EXTENDED 输出的 schema 变化

- **说明：**
SHOW TABLE EXTENDED的输出schema从database: string变成了namespace: string。
 - **Spark2.4.x:** SHOW TABLE EXTENDED的输出schema是database: string。
 - **Spark3.3.x:**
SHOW TABLE EXTENDED的输出schema从database: string变成了namespace: string。
其中对于内置catalog，namespace字段被命名为database；对于v2 catalog没有变化。
如果你希望在Spark 3.3.x版本中恢复到Spark 2.4.x版本的样式，可以通过将spark.sql.legacy.keepCommandOutputSchema设置为true来实现。
- **升级引擎版本后是否对作业有影响：**
有影响，请排查作业中与SHOW TABLES有关的使用方法，并按上述说明适配新版本的使用要求。
- **示例代码：**
执行sql:
show table extended like 'table%';

- Spark 2.4.5

database	tableName	isTemporary	information
db1	table1	false	Database:db1...
- Spark 3.3.1

namespace	tableName	isTemporary	information
db1	table1	false	Database:db1...

表刷新对依赖项缓存的影响

- **说明:**
升级Spark3.3.x版本后表刷新会清除表的缓存数据，但保持依赖项缓存。
- **Spark2.4.x:** 在Spark 2.4.x版本中，当执行表刷新操作（如REFRESH TABLE）时，不会保留依赖项（例如视图）的缓存数据。

```
ALTER TABLE .. ADD PARTITION
ALTER TABLE .. RENAME PARTITION
ALTER TABLE .. DROP PARTITION
ALTER TABLE .. RECOVER PARTITIONS
MSCK REPAIR TABLE
LOAD DATA
REFRESH TABLE
TRUNCATE TABLE
spark.catalog.refreshTable
```
- **Spark3.3.x:** 升级Spark3.3.x版本后表刷新会清除表的缓存数据，但保持依赖项缓存。
- **升级引擎版本后是否对作业有影响:**
升级新版本引擎后会增加原有依赖项的缓存数据。

表刷新对依赖该表的其他缓存操作的影响

- **说明:**
 - **Spark2.4.x:** Spark2.4.x中，刷新表时，只有当表本身被缓存时，才会触发引用该表的所有其他缓存的uncache操作。
 - **Spark3.3.x:** 升级新版本引擎后不管表本身是否有缓存，只要刷新表，那么依赖该表的其他缓存都会执行uncache操作。
- **升级引擎版本后是否对作业有影响:**
DLI功能增强，保证表刷新操作能对缓存生效，提高程序健壮性。

ADD PARTITION 新增支持使用类型化文字

- **说明:**
 - **Spark2.4.x:**
在Spark 2.4.x版本中，使用ADD PARTITION时，如果使用类型化文字（例如date'2020-01-01'），分区值会被解析为字符串值date'2020-01-01'，会生成一个非法的日期值，因此会添加一个值为null的分区。
正确的做法是使用字符串值，例如ADD PARTITION(dt = '2020-01-01')
 - **Spark3.3.x:** 在Spark 3.3.x版本中，对分区操作支持使用类型化文字，支持使用ADD PARTITION(dt = date'2020-01-01')，并且可以正确地将分区值解析为日期类型，而不是字符串。
- **升级引擎版本后是否对作业有影响:**
有影响，ADD PARTITION中对于类型化文字的处理方式的变化。
- **示例代码:**

准备数据:

```
create table test_part_type (id int,name string,pt date) PARTITIONED by (pt);
insert into test_part_type partition (pt = '2021-01-01') select 1,'name1';
insert into test_part_type partition (pt = date'2021-01-01') select 1,'name1';
```

执行sql:

```
select id,name,pt from test_part_type;
(配置参数spark.sql.forcePartitionPredicatesOnPartitionedTable.enabled为false)
```

- Spark 2.4.5
1 name1 2021-01-01
1 name1
- Spark 3.3.1
1 name1 2021-01-01
1 name1 2021-01-01

DayTimeIntervalType 的映射类型变化为 Duration

- **说明:**
在ArrowWriter和ArrowColumnVector开发者API中,从Spark 3.3.x版本开始,Spark SQL中的DayTimeIntervalType类型被映射到Apache Arrow的Duration类型。
 - **Spark2.4.x:** DayTimeIntervalType被映射到Apache Arrow的Interval类型。
 - **Spark3.3.x:** DayTimeIntervalType被映射到Apache Arrow的Duration类型。
- **升级引擎版本后是否对作业有影响:**
有影响, DayTimeIntervalType的映射类型发生变化。

日期差值返回结果的类型变化

- **说明:**
date减法表达式(如 date1 - date2)返回DayTimeIntervalType的值
 - **Spark2.4.x:** 返回CalendarIntervalType。
 - **Spark3.3.x:** 返回DayTimeIntervalType。
恢复之前的行为,将 spark.sql.legacy.interval.enabled 设置为 true
- **升级引擎版本后是否对作业有影响:**
有影响,日期差值返回结果默认类型变化。

单位到单位间隔的映射类型的变化

- **说明:**
 - **Spark2.4.x:** Spark2.4.x版本中单位到单位的间隔(如 INTERVAL '1-1' YEAR TO MONTH)和单位列表间隔(如 INTERVAL '3' DAYS '1' HOUR)将转换为CalendarIntervalType类型。
 - **Spark3.3.x:** 在Spark3.3.x版本中,单位到单位的间隔(如 INTERVAL '1-1' YEAR TO MONTH)和单位列表间隔(如 INTERVAL '3' DAYS '1' HOUR)将转换为ANSI间隔类型:YearMonthIntervalType或DayTimeIntervalType类型。
在Spark3.3.x版本中如果希望恢复到Spark2.4.x之前的映射类型,可以通过设置配置项 spark.sql.legacy.interval.enabled为true来实现。

- **升级引擎版本后是否对作业有影响：**
有影响，映射后的数据类型发生变化。

timestamps 减法表达式返回值类型变化

- **说明：**
 - **Spark2.4.x:** timestamps减法表达式，如select timestamp'2021-03-31 23:48:00'- timestamp'2021-01-01 00:00:00'返回CalendarIntervalType类型的值。
 - **Spark3.3.x:** timestamps减法表达式，如select timestamp'2021-03-31 23:48:00'- timestamp'2021-01-01 00:00:00'返回DayTimeIntervalType类型的值。
在Spark3.3.x版本中如果希望恢复到Spark2.4.x之前的映射类型，可以通过设置配置项spark.sql.legacy.interval.enabled为true来实现。
- **升级引擎版本后是否对作业有影响：**
有影响，映射后的数据类型发生变化。

不再支持混合使用年月字段和日时间字段

- **说明：**
 - **Spark2.4.x:** 单位列表间隔文字可以混合使用年月字段（YEAR 和 MONTH）和日时间字段（WEEK、DAY、...、MICROSECOND）。
 - **Spark3.3.x:** 单位列表间隔文字不能混合使用年月字段（YEAR 和 MONTH）和日时间字段（WEEK、DAY、...、MICROSECOND）。提示无效输入。
在Spark3.3.x版本中如果希望恢复到Spark2.4.x之前的使用方式，可以通过设置配置项spark.sql.legacy.interval.enabled为true来实现。
- **升级引擎版本后是否对作业有影响：**
有影响。

CREATE TABLE .. LIKE .. 命令不能使用保留属性

- **说明：**

CREATE TABLE .. LIKE .. 命令不能使用保留属性

 - **Spark2.4.x:** Spark2.4.x版本中在执行CREATE TABLE .. LIKE .. 命令时可以使用保留属性。
例如：TBLPROPERTIES('location'='/tmp') 不会改变表的位置但会创建一个无效属性。
 - **Spark3.3.x:** Spark3.3.x版本中CREATE TABLE .. LIKE .. 命令不能使用保留属性，
例如使用TBLPROPERTIES('location'='/tmp')或
TBLPROPERTIES('owner'='yao')会直接失败。
- **升级引擎版本后是否对作业有影响：**
有影响。
- **示例代码1：**

准备数据：

```
CREATE TABLE test0(id int, name string);  
CREATE TABLE test_like_properties LIKE test0 LOCATION 'obs://bucket1/test/test_like_properties';
```

执行sql:

```
DESCRIBE FORMATTED test_like_properties;
```

- Spark 2.4.5
正常显示location
- Spark 3.3.1
正常显示location

- **示例代码2:**

准备数据:

```
CREATE TABLE test_like_properties0(id int) using parquet LOCATION 'obs://bucket1/dbgms/  
test_like_properties0';  
CREATE TABLE test_like_properties1 like test_like_properties0 tblproperties('location'='obs://bucket1/  
dbgms/test_like_properties1');
```

执行sql:

```
DESCRIBE FORMATTED test_like_properties1;
```

- Spark 2.4.5
DLI.0005:
mismatched input 'tblproperties' expecting {<EOF>, 'LOCATION'}
- Spark 3.3.1
The feature is not supported: location is a reserved table property, please use the LOCATION clause to specify it.

包含自动生成的别名时创建视图失败

- **说明:**

- **Spark2.4.x:** Spark2.4.x版本中如果语句中包含自动生成的别名，则正常执行且无提示信息。
- **Spark3.3.x:** Spark3.3.x版本中如果语句中包含自动生成的别名，则创建/更改视图将失败。

在Spark3.3.x版本中如果希望恢复到Spark2.4.x之前的使用方式，可以通过设置配置项spark.sql.legacy.allowAutoGeneratedAliasForView为true来实现。

- **升级引擎版本后是否对作业有影响:**

有影响。

- **示例代码:**

准备数据:

```
create table test_view_alis(id1 int,id2 int);  
INSERT INTO test_view_alis VALUES(1,2);
```

执行sql:

```
create view view_alis as select id1 + id2 from test_view_alis;
```

- Spark 2.4.5
执行成功
- Spark 3.3.1
报错
Not allowed to create a permanent view `view_alis` without explicitly assigning an alias for expression (id1 + id2)

如果在Spark 3.3.1版本添加如下参数后，执行SQL成功。

```
spark.sql.legacy.allowAutoGeneratedAliasForView = true
```

日期加减时间字段间隔后的返回值类型的变化

- **说明:**

date +/-只有日期时间字段（如date'2011-11-11'）的间隔+间隔12小时返回类型变化。

 - **Spark2.4.x:** 在Spark 2.4.x中，当对定义为FloatType或DoubleType的JSON属性进行日期加减操作时，例如date'2011-11-11'加上或减去一个时间间隔（如12小时），返回的类型是日期（DateType）。
 - **Spark3.3.x:** Spark 3.3.x版本中，对于同样的操作，返回的类型变为时间戳（TimestampType），用于保持与Hive的兼容性。
- **升级引擎版本后是否对作业有影响:**

有影响。
- **示例代码:**

执行sql:

```
select date '2011-11-11' - interval 12 hour
```

 - Spark 2.4.5
2011-11-10
 - Spark 3.3.1
1320897600000

Spark SQL 支持 Char/Varchar 类型

- **说明:**
 - **Spark2.4.x:** 在Spark2.4.x版本中，Spark SQL表字段不支持Char/Varchar类型，当指定为Char/Varchar类型时会强制转换为String类型。
 - **Spark3.3.x:** Spark SQL表字段支持CHAR/CHARACTER和VARCHAR类型。
- **升级引擎版本后是否对作业有影响:**

无影响。
- **示例代码:**

准备数据:

```
create table test_char(id int,name varchar(24),name2 char(24));
```

执行sql:

```
show create table test_char;
```

 - Spark 2.4.5
create table `test_char`(`id` INT,`name` STRING,`name2` STRING)
ROW FORMAT...
 - Spark 3.3.1
create table test_char(id INT,name VARCHAR(24),name2 VARCHAR(24))
ROW FORMAT...

空值分区的查询语句不同

- **说明:**
 - **Spark2.4.x:**

Spark 3.0.1或更早版本中，如果分区列是字符串类型，则将其解析为其文本表示形式的字符串文本，例如字符串“null”。

通过part_col='null'查询空值分区的数据。
 - **Spark3.3.x:**

`PARTITION(col=null)`始终在分区规范中解析为null，即使分区列是字符串类型。

通过part_col is null查询空值分区的数据。

- **升级引擎版本后是否对作业有影响：**

有影响，涉及对空值分区的查询需要适配

- **示例代码：**

准备数据：

```
CREATE TABLE test_part_null (col1 INT, p1 STRING) USING PARQUET PARTITIONED BY (p1);  
INSERT INTO TABLE test_part_null PARTITION (p1 = null) SELECT 0;
```

执行sql：

```
select * from test_part_null;
```

- Spark 2.4.5
0 null

Spark 2.4.5版本执行select * from test_part_null where p1='null'可查到分区数据。

- Spark 3.3.1
0

Spark 3.3.1版本执行select * from test_part_null where p1 is null才可查询到数据。

对分区表数据的处理方式不同

- **说明：**

datasourcev1分区外表，路径下已经存在不带uuid的分区路径数据。

执行insert overwrite partition操作，Spark3.3.x会清除之前不带uuid的分区数据，Spark2.4.x不会清理。

- **Spark2.4.x：**
保留不带uuid分区路径下数据。

- **Spark3.3.x：**
会删除不带uuid分区路径下数据。

- **升级引擎版本后是否对作业有影响：**

有影响，会清理脏数据。

- **示例代码：**

准备数据：

obs://bucket1/test/overwrite_datasource下创建pt=pt1目录，并导入一个parquet数据文件。

```
create table overwrite_datasource(id int,name string,pt string) using parquet PARTITIONED by(pt)  
LOCATION 'obs://bucket1/test/overwrite_datasource';  
SELECT * FROM overwrite_datasource1 where pt='pt1'两个版本均查询不到数据。
```

执行sql：

```
insert OVERWRITE table overwrite_datasource partition(pt='pt1') values(2,'aa2');
```

- Spark 2.4.5
保留了pt=pt1目录。

- Spark 3.3.1
删除了pt=pt1目录。

导出 CSV 文件时保留特殊字符的引号

- **说明:**
 - **Spark2.4.x:**

在Spark 2.4.x版本中，导出CSV文件时，如果字段值中包含特殊字符如换行符（\n）和回车符（\r），并且这些特殊字符被引号（例如双引号）包围，Spark会自动处理这些引号，在导出的CSV文件中舍弃这些引号。

例如，字段值"a\rb"在导出时不会包含引号。
 - **Spark3.3.x:**

在Spark 3.3.x版本中，优化了对于CSV文件的导出处理，如果字段值中包含特殊字符，并且这些特殊字符被引号包围，Spark会保留这些引号。

例如：字段值"a\rb"在导出时，引号仍被保留在最终的CSV文件中。

- **升级引擎版本后是否对作业有影响:**

对查询结果无影响，但导出文件样式有影响。

- **示例代码:**

准备数据:

```
create table test_null2(str1 string,str2 string,str3 string,str4 string);
insert into test_null2 select "a\rb", null, "1\n2", "ab";
```

执行sql:

```
SELECT * FROM test_null2;
```

- Spark 2.4.5
a b 1 2 ab

- Spark 3.3.1
a b 1 2 ab

导出查询结果到obs，查看csv文件内容:

- Spark 2.4.5
a
b,"";"1
2",ab

- Spark 3.3.1
"a
b",,"1
2",ab

新增支持自适应 Skip partial agg 功能的配置

- **说明:**

Spark3.3.x版本中新增支持自适应Skip partial agg功能，即当Partial Agg效果不佳时，可以直接跳过，避免Partial Agg带来的额外性能消耗。相关参数:

 - spark.sql.aggregate.adaptivePartialAggregationEnabled: 用于控制是否启用自适应Skip partial agg功能。当设置为true时，Spark会根据运行时的统计数据动态决定是否跳过部分聚合。
 - spark.sql.aggregate.adaptivePartialAggregationInterval: 该参数用于配置分析间隔，即在处理了多少行数据之后，Spark会进行一次分析，用来决定是否跳过部分聚合。
 - spark.sql.aggregate.adaptivePartialAggregationRatio: 该参数是判断是否跳过的阈值，用于判断“已处理的group分组/已处理的行数”的比例。如果这个比例大于配置的阈值，则认为预聚合效果不好，此时Spark可以选择跳过部分聚合，避免进一步的性能损失。

在使用时系统先按照spark.sql.aggregate.adaptivePartialAggregationInterval配置的间隔进行分析，当处理的行数到达间隔之后，再计算”已处理的group分组/已处理的行数”，如果比例大于配置的阈值，则认为预聚合效果不好，此时可以直接选择跳过。

- **升级引擎版本后是否对作业有影响：**

DLI功能增强。

新增支持 Parallel Multi-Insert

- **说明：**

Spark3.3.x版本中新增支持Parallel Multi-Insert，如果SQL存在multi-insert的场景，在同一个SQL里插入到多个表中，这类SQL在Spark开源本身是串行处理的，性能受到制约。针对这类SQL，Spark3.3.x版本中DLI新增支持multi-insert并行化的优化处理，可以让所有的insert都并发执行，提升处理性能。

在使用时需配置开启以下功能开关（默认关闭）：

```
spark.sql.lazyExecutionForDDL.enabled=true
```

```
spark.sql.parallelMultiInsert.enabled=true
```

- **升级引擎版本后是否对作业有影响：**

DLI功能增强，增强multi-insert并行化特性，提升作业运行的可靠性。

新增支持 Enhance Reuse Exchange

- **说明：**

Spark3.3.x版本中新增支持Enhance Reuse Exchange，SQL的对应plan存在sort merge join可重用的条件，通过打开相应开关spark.sql.execution.enhanceReuseExchange.enabled，可以实现SMJ plan node重用。

在使用时需配置开启以下功能开关（默认关闭）：

```
spark.sql.execution.enhanceReuseExchange.enabled=true
```

- **升级引擎版本后是否对作业有影响：**

DLI功能增强。

TIMESTAMP 类型字段读取差异

- **说明：**

TIMESTAMP类型字段读取差异，对于Asia/Shanghai时区，时间在1900-01-01 08:05:43之前的值，Spark 2.4.5版本写入后Spark 3.3.1版本读取值与Spark 2.4.5版本读取值不同。

- **Spark2.4.x：**

以Asia/Shanghai时区的 1900-01-01 00:00:00 为例，通过Spark 2.4.5版本队列写入，Spark 2.4.5版本读取后得到的值为 -2209017600000。

- **Spark3.3.x：**

以Asia/Shanghai时区的 1900-01-01 00:00:00 为例，通过Spark 2.4.5版本队列写入，Spark 3.3.1版本配置 **spark.sql.parquet.int96RebaseModeInRead=LEGACY**，读取后得到的值为 -2209017943000。

- **升级引擎版本后是否对作业有影响：**

有影响，需要评估TIMESTAMP类型字段的使用方式。

- **示例代码：**

在SQL界面配置：

```
spark.sql.session.timeZone=Asia/Shanghai
```

- **Spark 2.4.5**

```
create table parquet_timestamp_test (id int, col0 string, col1 timestamp) using parquet;  
insert into parquet_timestamp_test values (1, "245", "1900-01-01 00:00:00");
```

执行SQL读取数据：

```
select * from parquet_timestamp_test;
```

查询结果：

id	col0	col1
1	245	-2209017600000

- **Spark 3.3.1**

```
spark.sql.parquet.int96RebaseModelInRead=LEGACY
```

执行SQL读取数据：

```
select * from parquet_timestamp_test;
```

查询结果：

id	col0	col1
1	245	-2209017943000

- **配置项说明：**

这些配置项提供了Spark如何处理DATE, TIMESTAMP类型字段中特定的时间（在外推格里历和儒略历之间有争议的时间），例如：对于Asia/Shanghai时区，指的是在1900-01-01 08:05:43之前的值的处理方式。

- [Datasource parquet表配置项说明](#)

表 2-4 Spark 3.3.1 Datasource parquet 表配置项

配置项	默认值	描述
spark.sql.parquet.int96RebaseModelInRead	LEGACY (Spark SQL 默认配置)	<p>读取Parquet文件中INT96类型的TIMESTAMP字段时生效。</p> <ul style="list-style-type: none"> ● EXCEPTION: 遇到特定的时间会抛出报错，读取操作将失败。 ● CORRECTED: Spark 不会进行重新调整，而是按照原样读取日期/时间戳。 ● LEGACY: Spark 会将日期/时间戳从传统的混合模式（儒略历 + 格里历）日历重新调整到外推格里历。 <p>该配置项仅在 Parquet 文件的写入信息（如 Spark、Hive）未知时才生效。</p>
spark.sql.parquet.int96RebaseModelInWrite	LEGACY (Spark SQL 默认配置)	<p>写入Parquet文件中INT96类型的TIMESTAMP字段时生效。</p> <ul style="list-style-type: none"> ● EXCEPTION: 遇到特定的时间会抛出报错，写入操作将失败 ● CORRECTED: Spark 不会进行重新调整，而是按照原样写入日期/时间戳 ● LEGACY: 当写入 Parquet 文件时，Spark 会将日期/时间戳从外推格里历重新调整到传统的混合模式（儒略历 + 格里历）。

配置项	默认值	描述
spark.sql.parquet.date timeRebaseModelInRead	LEGACY (Spark SQL 默认配置)	<p>读取DATE、TIMESTAMP_MILLIS、TIMESTAMP_MICROS逻辑类型的字段生效。</p> <ul style="list-style-type: none"> ● EXCEPTION: 遇到特定的时间会抛出报错，读取操作将失败。 ● CORRECTED: Spark 不会进行重新调整，而是按照原样读取日期/时间戳。 ● LEGACY: Spark 会将日期/时间戳从传统的混合模式（儒略历 + 格里历）日历重新调整到外推格里历。 <p>该配置项仅在 Parquet 文件的写入信息（如 Spark、Hive）未知时才生效。</p>
spark.sql.parquet.date timeRebaseModelInWrite	LEGACY (Spark SQL 默认配置)	<p>写入DATE、TIMESTAMP_MILLIS、TIMESTAMP_MICROS逻辑类型的字段生效。</p> <ul style="list-style-type: none"> ● EXCEPTION: 遇到特定的时间会抛出报错，写入操作将失败 ● CORRECTED: Spark 不会进行重新调整，而是按照原样写入日期/时间戳 ● LEGACY: 当写入 Parquet 文件时，Spark 会将日期/时间戳从外推格里历重新调整到传统的混合模式（儒略历 + 格里历）。

- [Datasource avro表配置项说明](#)

表 2-5 Spark 3.3.1 Datasource avro 表配置项

配置项	默认值	描述
spark.sql.avro.datetimeRebaseModeInRead	LEGACY (Spark SQL 默认配置)	<p>读取DATE、TIMESTAMP_MILLIS、TIMESTAMP_MICROS逻辑类型的字段生效。</p> <ul style="list-style-type: none"> ● EXCEPTION: 遇到特定的时间会抛出报错，读取操作将失败。 ● CORRECTED: Spark 不会进行重新调整，而是按照原样读取日期/时间戳。 ● LEGACY: Spark 会将日期/时间戳从传统的混合模式（儒略历 + 格里历）日历重新调整到外推格里历。 <p>该配置项仅在 Avro 文件的写入信息（如 Spark、Hive）未知时才生效。</p>
spark.sql.avro.datetimeRebaseModeInWrite	LEGACY (Spark SQL 默认配置)	<p>写入DATE、TIMESTAMP_MILLIS、TIMESTAMP_MICROS逻辑类型的字段生效。</p> <ul style="list-style-type: none"> ● EXCEPTION: 遇到特定的时间会抛出报错，写入操作将失败 ● CORRECTED: Spark 不会进行重新调整，而是按照原样写入日期/时间戳 ● LEGACY: 当写入 Avro 文件时，Spark 会将日期/时间戳从外推格里历重新调整到传统的混合模式（儒略历 + 格里历）。

from_unixtime 函数差异

- **说明:**
 - **Spark2.4.x:**
以Asia/Shanghai时区的 -2209017600 为例，返回值为1900-01-01 00:00:00。
 - **Spark3.3.x:**
以Asia/Shanghai时区的 -2209017943 为例，返回值为 1900-01-01 00:00:00。
- **升级引擎版本后是否对作业有影响:**
有影响，需要检查使用该函数的场景。
- **示例代码:**
在SQL界面配置:
spark.sql.session.timeZone=Asia/Shanghai
 - Spark 2.4.5
执行SQL读取数据:

```
select from_unixtime(-2209017600);
```


查询结果:
1900-01-01 00:00:00
 - Spark 3.3.1
执行SQL读取数据:

```
select from_unixtime(-2209017600);
```


查询结果
1900-01-01 00:05:43

unix_timestamp 函数差异

- **说明:**
对于Asia/Shanghai时区，小于1900-01-01 08:05:43的值。
 - **Spark2.4.x:**
以Asia/Shanghai时区的 1900-01-01 00:00:00 为例，返回值为 -2209017600。
 - **Spark3.3.x:**
以Asia/Shanghai时区的 1900-01-01 00:00:00 为例，返回值为 -2209017943。
- **升级引擎版本后是否对作业有影响:**
有影响，需要检查使用该函数的场景。
- **示例代码:**
在SQL界面配置:
spark.sql.session.timeZone=Asia/Shanghai
 - Spark 2.4.5
执行SQL读取数据:

```
select unix_timestamp('1900-01-01 00:00:00');
```


查询结果:
-2209017600

- Spark 3.3.1

执行SQL读取数据：

```
select unix_timestamp('1900-01-01 00:00:00');
```

查询结果

```
-2209017943
```

2.7.2 Spark 2.4.x 与 Spark 3.3.x 版本在通用队列的差异对比

DLI整理了Spark2.4.x与Spark3.3.x版本在通用队列的差异，便于您了解Spark版本升级后通用队列上运行的作业在适配新版本引擎时的影响。

log4j 依赖从 1.x 版本修改为 2.x 版本

- 说明：
 - log4j依赖从1.x版本修改为2.x版本
 - **Spark2.4.x**：log4j依赖1.x版本（社区不再支持）。
 - **Spark3.3.x**：log4j依赖2.x版本。
- 升级引擎版本后是否对作业有影响：
有影响

Spark3.3.x 不支持 v1 表

- 说明：
 - Spark2.4.x支持datasourcev1、datasourcev2表。Spark3.3.x不支持v1表。具体说明请参考[DLI datasourceV1表和datasourceV2表](#)。
 - **Spark2.4.x**：支持datasourcev1、datasourcev2表。
 - **Spark3.3.x**：不支持支持datasourcev1表。
- 升级引擎版本后是否对作业有影响：
有影响，建议在Spark 2.4.5版本整改到v2表后再升级Spark 3.3.1，具体操作指导可以参考[DLI datasourceV1表和datasourceV2表](#)的中的示例。

默认情况下空的 input split 不创建 partition

- 说明：
 - **Spark2.4.x**：默认情况下空的input split将创建partition。
 - **Spark3.3.x**：默认情况下空的input split不创建partition。
Spark3.3.x时spark.hadoopRDD.ignoreEmptySplits=true。
- 升级引擎版本后是否对作业有影响：
有影响，需要判断是否使用分区名做业务判断。

eventlog 的压缩格式设置为 zstd

- 说明：
 - Spark3.3.x版本中，spark.eventLog.compression.codec的默认值被设置为zstd，Spark在压缩事件日志时将不再支持使用spark.io.compression.codec的参数值。
 - **Spark2.4.x**：使用spark.io.compression.codec的参数值作为eventlog的压缩格式。

- **Spark3.3.x:** spark.eventLog.compression.codec默认设置为zstd。
- **升级引擎版本后是否对作业有影响:**
有影响，eventlog的压缩格式发生变化。

spark.launcher.childConectionTimeout 修改

- **说明:**
 - **Spark2.4.x:** 配置名为spark.launcher.childConectionTimeout
 - **Spark3.3.x:** 配置名修改为spark.launcher.childConnectionTimeout
- **升级引擎版本后是否对作业有影响:**
有影响，配置参数名称变化。

Spark3.3.x 不再支持将 Apache Mesos 作为资源管理器

- **说明:**
 - **Spark2.4.x:** Spark 2.4.x版本中使用Apache Mesos作为资源管理器。
 - **Spark3.3.x:** Spark3.3.x不再支持将Apache Mesos作为资源管理器。
- **升级引擎版本后是否对作业有影响:**
功能增强，Spark 2.4.x版本中使用Mesos作为资源管理器，升级到Spark 3.3.x后，你需要考虑切换到其他资源管理器。

Spark3.3.x 会在应用程序自行终止时删除 K8s driver

- **说明:** Spark3.3.x会在应用程序自行终止时删除K8s driver。
- **升级引擎版本后是否对作业有影响:**
功能增强，升级到Spark 3.3.x后，对于之前依赖于Kubernetes作为资源管理器的作业会有影响。Spark 3.3.x在应用程序终止时会自动删除driver pod可能会影响到作业的资源管理和清理流程。

Spark3.3.x 支持自定义 k8s 的调度器

- **说明:**
 - **Spark2.4.x:** 不支持使用指定Kubernetes调度器来管理Spark作业的资源分配和调度。
 - **Spark3.3.x:** Spark3.3.x支持自定义k8s的调度器。
- **升级引擎版本后是否对作业有影响:**
功能增强，支持自定义调度器管理资源的分配和调度。

Spark 将不可为 null 的模式转换为可空

- **说明:**

在Spark 2.4.x版本中，当用户指定的schema包含不可为空的字段时，Spark会将这些不可为null的模式转换为可空的。

但是在Spark 3.3.x版本中，Spark尊重用户指定的schema中的nullability，即如果字段被定义为不可为null，Spark会保持该配置要求，不会自动转换为可空的字段。

 - **Spark2.4.x:** 在Spark 2.4.x版本中，当用户指定的schema包含不可为空的字段时，Spark会将这些不可为null的模式转换为可空的。

- **Spark3.3.x:** Spark不会自动转换为可空的字段。
如果希望在Spark 3.3.x版本中恢复到Spark 2.4.x版本的执行方式，您可以通过将 `spark.sql.legacy.respectNullabilityInTextDatasetConversion` 设置为 `true` 来实现。
- **升级引擎版本后是否对作业有影响:**
有影响。
- **示例代码:**
执行sql:

```
spark.read.schema(StructType(  
  StructField("f1", LongType, nullable = false) ::  
  StructField("f2", LongType, nullable = false) :: Nil)  
) .option("mode", "DROPMALFORMED").json(Seq("""{"f1": 1}""").toDS).show(false);
```
- **Spark 2.4.5**

```
|f1 |f2 |  
+---+---+  
|1  |0  |
```
- **Spark 3.3.1**

```
|f1 |f2 |  
+---+---+  
|1  |null|
```

Spark scala 版本变更

- **说明:**
Spark scala版本变更。
 - **Spark2.4.x:** Spark scala版本为2.11。
 - **Spark3.3.x:** Spark scala版本升级到2.12。
- **升级引擎版本后是否对作业有影响:**
有影响，jar需要升级scala版本编译。

PySpark 支持 python 版本变更

- **说明:**
PySpark支持python版本变更。
 - **Spark2.4.x:** PySpark支持python版本范围2.6+版本到3.7+版本。
 - **Spark3.3.x:** PySpark支持Python版本范围3.6及以上版本。
- **升级引擎版本后是否对作业有影响:**
依赖版本变化，有影响，需要排查是否涉及。

PySpark-pandas 支持版本变更

- **说明:**
 - **Spark2.4.x:** 在Spark 2.4.x版本中，PySpark并没有要求指定Pandas的版本。
 - **Spark3.3.x:** 从Spark 3.3.x版本开始，PySpark需要0.23.2或更高版本的pandas才能使用pandas相关功能，如`toPandas`、`createDataFrame from pandas DataFrame`等。
- **升级引擎版本后是否对作业有影响:**
依赖版本变化，有影响，需要排查是否涉及。

PySpark-PyArrow 支持版本变更

- **说明:**
 - **Spark2.4.x:** 在Spark 2.4.x版本中, PySpark并没有要求指定PyArrow的版本。
 - **Spark3.3.x:** 从Spark 3.3.x版本开始, PySpark需要0.12.1或更高版本的PyArrow才能使用PyArrow相关功能, 如Pandas_udf、toPandas等。
- **升级引擎版本后是否对作业有影响:**
依赖版本变化, 有影响, 需要排查是否涉及。

以 command 命名 DataFrameWriter 触发的查询

在Spark 3.2.x版本中, 当DataFrameWriter触发的查询执行被发送给QueryExecutionListener时, 这些查询的名称总是被设置为command。而在Spark 3.1及更早版本中, 这些查询的名称可能是save、insertInto或saveAsTable之一, 这取决于具体的操作。

- **说明:**
DataFrameWriter触发的查询执行在发送到QueryExecutionListener时, 始终以command命名
 - **Spark2.4.x:** 名称为save、insertInto、saveAsTable中的一个
 - **Spark3.3.x:** command命名
- **升级引擎版本后是否对作业有影响:**
有影响

DATE、TIMESTAMP 类型字段读取差异

- **说明:**
DATE、TIMESTAMP类型字段读取差异, 对于Asia/Shanghai时区, 时间在1900-01-01 08:05:43之前的值, Spark 2.4.5版本写入后Spark 3.3.1版本读取值与Spark 2.4.5版本读取值不同。
 - **Spark2.4.x:**
以Asia/Shanghai时区的 1900-01-01 00:00:00 为例, 通过Spark 2.4.5版本队列写入, Spark 2.4.5版本读取后得到的值为 1900-01-01 00:00:00。
 - **Spark3.3.x:**
以Asia/Shanghai时区的 1900-01-01 00:00:00 为例, 通过Spark 2.4.5版本队列写入, Spark 3.3.1版本配置 **spark.sql.parquet.int96RebaseModelInRead=LEGACY**, 读取后得到的值为 1900-01-01 00:00:00, 但是配置 **spark.sql.parquet.int96RebaseModelInRead=CORRECTED**时, 读取后得到的值为1900-01-01 00:05:43。
- **升级引擎版本后是否对作业有影响:**
有影响, 需要评估DATE、TIMESTAMP类型字段的使用方式。
- **示例代码:**
在作业中配置:

```
spark.sql.session.timeZone=Asia/Shanghai
```

 - Spark 2.4.5

```
spark.sql("create table parquet_timestamp_test (id int, col0 string, col1 timestamp) using
parquet");
spark.sql("insert into parquet_timestamp_test values (1, '245', '1900-01-01 00:00:00')");
```

执行SQL读取数据:

```
spark.sql("select * from parquet_timestamp_test").show();
```

查询结果:

```
+---+-----+-----+
| id|col0|          col1|
+---+-----+-----+
| 1| 245|1900-01-01 00:00:00|
+---+-----+-----+
```

- Spark 3.3.1

```
spark.sql.parquet.int96RebaseModelInRead=LEGACY
```

执行作业读取数据:

```
spark.sql("select * from parquet_timestamp_test").show();
```

查询结果

```
+---+-----+-----+
| id|col0|          col1|
+---+-----+-----+
| 1| 245|1900-01-01 00:00:00|
+---+-----+-----+
```

可以再更换配置:

```
spark.sql.parquet.int96RebaseModelInRead=CORRECTED
```

并重新执行SQL进行读取:

```
spark.sql("select * from parquet_timestamp_test").show();
```

查询结果:

```
+---+-----+-----+
| id|col0|          col1|
+---+-----+-----+
| 1| 245|1900-01-01 00:05:43|
+---+-----+-----+
```

- **配置项说明:**

这些配置项提供了Spark如何处理DATE, TIMESTAMP类型字段中特定的时间（在外推格里历和儒略历之间有争议的时间），例如：对于Asia/Shanghai时区，指的是在1900-01-01 08:05:43之前的值的处理方式。

- [Datasource parquet表配置项说明](#)

表 2-6 Spark 3.3.1 Datasource parquet 表配置项

配置项	默认值	描述
spark.sql.parquet.int96RebaseModelInRead	EXCEPTION (Spark作业默认配置)	<p>读取Parquet文件中INT96类型的TIMESTAMP字段时生效。</p> <ul style="list-style-type: none"> ● EXCEPTION: 遇到特定的时间会抛出报错，读取操作将失败。 ● CORRECTED: Spark 不会进行重新调整，而是按照原样读取日期/时间戳。 ● LEGACY: Spark 会将日期/时间戳从传统的混合模式 (儒略历 + 格里历) 日历重新调整到外推格里历。 <p>该配置项仅在 Parquet 文件的写入信息 (如 Spark、Hive) 未知时才生效。</p>
spark.sql.parquet.int96RebaseModelInWrite	EXCEPTION (Spark作业默认配置)	<p>写入Parquet文件中INT96类型的TIMESTAMP字段时生效。</p> <ul style="list-style-type: none"> ● EXCEPTION: 遇到特定的时间会抛出报错，写入操作将失败 ● CORRECTED: Spark 不会进行重新调整，而是按照原样写入日期/时间戳 ● LEGACY: 当写入 Parquet 文件时，Spark 会将日期/时间戳从外推格里历重新调整到传统的混合模式 (儒略历 + 格里历) 。

配置项	默认值	描述
spark.sql.parquet.date timeRebaseModelInRead	EXCEPTION (Spark作 业默认配置)	<p>读取DATE、TIMESTAMP_MILLIS、TIMESTAMP_MICROS逻辑类型的字段生效。</p> <ul style="list-style-type: none"> ● EXCEPTION: 遇到特定的时间会抛出报错，读取操作将失败。 ● CORRECTED: Spark 不会进行重新调整，而是按照原样读取日期/时间戳。 ● LEGACY: Spark 会将日期/时间戳从传统的混合模式（儒略历 + 格里历）日历重新调整到外推格里历。 <p>该配置项仅在 Parquet 文件的写入信息（如 Spark、Hive）未知时才生效。</p>
spark.sql.parquet.date timeRebaseModelInWrite	EXCEPTION (Spark作 业默认配置)	<p>写入DATE、TIMESTAMP_MILLIS、TIMESTAMP_MICROS逻辑类型的字段生效。</p> <ul style="list-style-type: none"> ● EXCEPTION: 遇到特定的时间会抛出报错，写入操作将失败 ● CORRECTED: Spark 不会进行重新调整，而是按照原样写入日期/时间戳 ● LEGACY: 当写入 Parquet 文件时，Spark 会将日期/时间戳从外推格里历重新调整到传统的混合模式（儒略历 + 格里历）。

- [Datasource avro表配置项说明](#)

表 2-7 Spark 3.3.1 Datasource avro 表配置项

配置项	默认值	描述
spark.sql.avro.datetim eRebaseModelnRead	EXCEPTION (Spark作 业默认配置)	<p>读取DATE、TIMESTAMP_MILLIS、TIMESTAMP_MICROS逻辑类型的字段生效。</p> <ul style="list-style-type: none"> ● EXCEPTION: 遇到特定的时间会抛出报错，读取操作将失败。 ● CORRECTED: Spark 不会进行重新调整，而是按照原样读取日期/时间戳。 ● LEGACY: Spark 会将日期/时间戳从传统的混合模式（儒略历 + 格里历）日历重新调整到外推格里历。 <p>该配置项仅在 Avro 文件的写入信息（如 Spark、Hive）未知时才生效。</p>
spark.sql.avro.datetim eRebaseModelnWrite	EXCEPTION (Spark作 业默认配置)	<p>写入DATE、TIMESTAMP_MILLIS、TIMESTAMP_MICROS逻辑类型的字段生效。</p> <ul style="list-style-type: none"> ● EXCEPTION: 遇到特定的时间会抛出报错，写入操作将失败 ● CORRECTED: Spark 不会进行重新调整，而是按照原样写入日期/时间戳 ● LEGACY: 当写入 Avro 文件时，Spark 会将日期/时间戳从外推格里历重新调整到传统的混合模式（儒略历 + 格里历）。

from_unixtime 函数差异

- **说明:**
 - **Spark2.4.x:**
以Asia/Shanghai时区的 -2209017600 为例，返回值为1900-01-01 00:00:00。
 - **Spark3.3.x:**
以Asia/Shanghai时区的 -2209017943 为例，返回值为 1900-01-01 00:00:00。
- **升级引擎版本后是否对作业有影响:**
有影响，需要检查使用该函数的场景。
- **示例代码:**
在作业中配置:

```
spark.sql.session.timeZone=Asia/Shanghai
```

- **Spark 2.4.5**
执行以下语句读取数据:

```
select from_unixtime(-2209017600);
```

查询结果:

```
+-----+  
|from_unixtime(-2209017600, yyyy-MM-dd HH:mm:ss)|  
+-----+  
|                1900-01-01 00:00:00|  
+-----+
```

- **Spark 3.3.1**
执行以下语句读取数据:

```
select from_unixtime(-2209017600);
```

查询结果

```
+-----+  
|from_unixtime(-2209017600, yyyy-MM-dd HH:mm:ss)|  
+-----+  
|                1900-01-01 00:05:43|  
+-----+
```

unix_timestamp 函数差异

- **说明:**
对于Asia/Shanghai时区，小于1900-01-01 08:05:43的值。
 - **Spark2.4.x:**
以Asia/Shanghai时区的 1900-01-01 00:00:00 为例，返回值为 -2209017600。
 - **Spark3.3.x:**
以Asia/Shanghai时区的 1900-01-01 00:00:00 为例，返回值为 -2209017943。
- **升级引擎版本后是否对作业有影响:**
有影响，需要检查使用该函数的场景。
- **示例代码:**
在作业中配置:

```
spark.sql.session.timeZone=Asia/Shanghai
```

- Spark 2.4.5

执行以下语句读取数据：

```
select unix_timestamp('1900-01-01 00:00:00');
```

查询结果：

```
+-----+
|unix_timestamp(1900-01-01 00:00:00, yyyy-MM-dd HH:mm:ss)|
+-----+
|                                -2209017600|
+-----+
```

- Spark 3.3.1

执行以下语句读取数据：

```
select unix_timestamp('1900-01-01 00:00:00');
```

查询结果

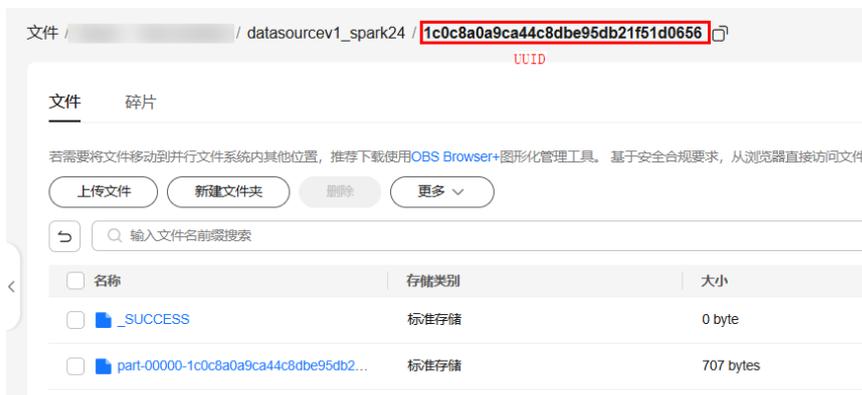
```
+-----+
|unix_timestamp(1900-01-01 00:00:00, yyyy-MM-dd HH:mm:ss)|
+-----+
|                                -2209017943|
+-----+
```

2.7.3 DLI datasourceV1 表和 datasourceV2 表

什么是 DLI datasourcev1 表和 DLI datasourcev2 表？

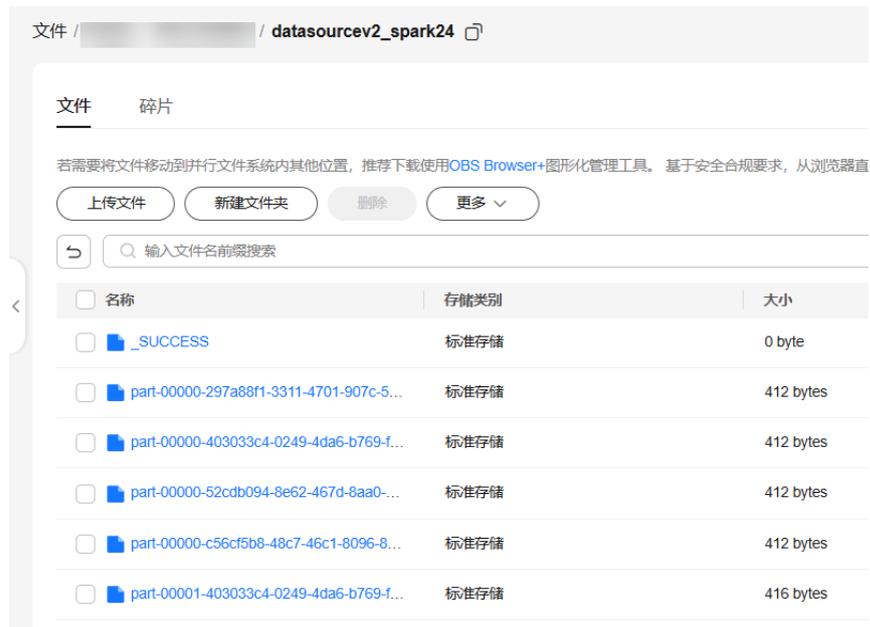
- DLI datasource v1表（以下简称V1表）：DLI的Datasource表格式，建表/插入/truncate命令使用DLI自定义的command，表的数据路径为\$tablepath/UUID/数据文件。

图 2-2 DLI datasource v1 表



- DLI datasource v2表（以下简称V2表）：spark开源的Datasource表，建表/插入/truncate命令使用spark开源的command，表的数据路径为\$tablepath/数据文件。

图 2-3 DLI datasource v2 表



DLI 各 Spark 版本对 V1、V2 表兼容性

表 2-8 DLI 各 Spark 版本对 V1、V2 表兼容列表

表类型	Spark 2.3 SQL队列	Spark 2.3 通用队列	Spark2.4 SQL队列	Spark2.4 通用队列	Spark 3.1 SQL队列	Spark 3.1 通用队列	Spark 3.3 SQL队列	Spark 3.3 通用队列
V1表	√	√	√	√	√	√	√	部分支持
V2表	×	×	√	√	×	×	√	√

表 2-9 Spark3.3 通用队列语法支持列表

表类型	select	create table	create table like	CTAS	insert into	insert overwrite	load data	alter table set location	truncate table
V1表	√	√	√	×	×	×	×	×	×
V2表	√	√	√	√	√	√	√	√	√

怎么确认当前用户创建的表是 v1 还是 v2 表？

1. 使用datasource语法建表：

```
CREATE TABLE IF NOT EXISTS table_name (id STRING) USING parquet;
```

2. 执行show create table查看TBLPROPERTIES下的"verison"字段的值

"v1"为V1表; "v2"则为V2表。

如需修改V1表为V2表请提交工单联系客户支持获取操作帮助。

升级示例

📖 说明

升级Spark引擎和修改数据表时，如新建队列时切换了计算资源类型可能会导致计费资源费用变化。

- 如果原队列使用的是弹性资源池类型的计算资源，那么新建队列不涉及计算资源费用变化。
- 如果原队列使用的是非弹性资源池类型的计算资源，那么使用弹性资源池资源新建队列后计算资源费用将发生变化，具体情况以计算资源的价格详情为准。
- **示例1：使用SQL队列，将Spark版本从Spark 2.4.x升级至Spark 3.3.1对数据表的版本有影响吗？**
不需要，Spark 2.4.x的SQL队列支持V1表和V2表，因此升级Spark版本只需要考虑Spark版本对SQL语法的兼容性。
- **示例2：使用通用队列，将Spark版本从Spark 2.4.x升级至Spark 3.3.1对数据表的版本有影响吗？**
Spark 2.4.x通用队列支持V1表和V2表，但Spark3.3.x通用队列不支持V1表。因此如需将Spark版本从Spark 2.4.x升级至Spark 3.3.1需经过以下步骤：
 - a. 将Spark 2.4.x的V1表修改为V2表。
 - b. 升级Spark 2.4.x的V2表升级为Spark 3.3.1的V2表。
同时还需考虑Spark jar作业API语法的兼容性。

表 2-10 DLI 各 Spark 版本对 V1、V2 表兼容列表

表类型	Spark2.4 通用队列	Spark3.3 通用队列
V1表	√	部分支持
V2表	√	√

- **示例3：使用通用队列，怎样将Spark 2.3.2的V1表升级为Spark 3.3.1的V2表？**
Spark 2.3.2通用队列不支持V2表，Spark 3.3.1通用队列不支持V1表：
 - a. 将Spark 2.3.2的V1表升级至Spark 2.4.5的V1表。
 - b. 将Spark 2.4.5的V1表修改为V2表。
 - c. 升级Spark 2.4.5的V2表升级为Spark 3.3.1的V2表。
同时还需考虑Spark jar作业API语法的兼容性。

表 2-11 DLI 各 Spark 版本对 V1、V2 表兼容列表

表类型	Spark2.3 通用队列	Spark2.4 通用队列	Spark3.3 通用队列
V1表	√	√	部分支持
V2表	×	√	√