随笔 - 5 文章 - 0 评论 - 4

时序数据库 Apache-IoTDB 源码解析之文件索引块（五）

上一章聊到 TsFile 的文件组成，以及数据块的详细介绍。详情请见：

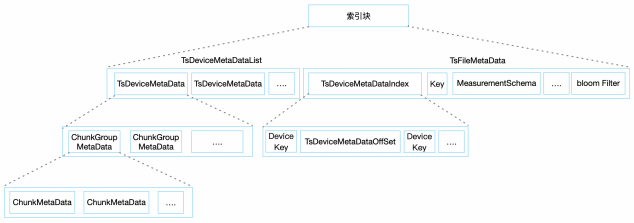
时序数据库 Apache-IoTDB 源码解析之文件数据块（四）

打一波广告，欢迎大家访问IoTDB 仓库，求一波Star。

这一章主要想聊聊：

1. TsFile索引块的组成
2. 索引块的查询过程
3. 索引块目前在做的改进项

索引块



索引块由两大部分组成，其写入的方式是从左到右写入，也就是从文件头向文件尾写入。但读出的方式是先读出TsFileMetaData 再读出TsDeviceMetaDataList 中的具体一部分。我们按照读取数据的顺序介绍：

TsFileMetaData

TsFileMetaData属于文件的 1 级索引，用来索引 Device 是否存在、在哪里等信息，其中主要保存了：

InfluxDB(2)
数据库性能(2)

随笔档案

2020年2月(5)

最新评论

1. Re:时序数据库 Apache-IoTDB 源码解析之文件数据块（四）
楼主您好，我刚接触这个数据库一天 安装的时候有个问题卡住了 能加个微信吗。指点一下

--Smile_灰太狼

2. Re:时序数据库 Apache-IoTDB 源码解析之文件数据块（四）
@daconglee 目前没有...

--Liutaohua

3. Re:时序数据库 Apache-IoTDB 源码解析之文件数据块（四）
有C#的读写代码吗?

--daconglee

4. Re:时序数据库 Apache-IoTDB 源码解析之前言（一）
有C#的读写代码吗?

--daconglee

阅读排行榜

1. 时序数据库 Apache-IoTDB 源码解析之文件索引块（五）(312)
2. 时序数据库 Apache-IoTDB 源码解析之系统架构（二）(227)
3. 时序数据库 Apache-IoTDB 源码解析之前言（一）(200)
4. 时序数据库 Apache-IoTDB 源码解析之文件数据块（四）(152)
5. 时序数据库 Apache-IoTDB 源码解析之

1. DeviceMetaDataIndexMap：Map结构，Key 是设备名，Value 是 TsDeviceMetaDataIndex ，保存了包含哪些 Device（逻辑概念上的一个集合一段时间内的数据，例如前几章我们讲到的：张三、李四、王五）以及他们的开始时间及结束时间、在左侧 TsDeviceMetaDataList 文件块中的偏移量等。
2. MeasurementSchemaMap：Map结构，Key 是测点的一个全路径，Value 是 measurementSchema ，保存了包含的测点数据(逻辑概念上的某一类数据的集合，如体温数据)的原信息，如：压缩方式，数据类型，编码方式等。
3. 最后是一个布隆过滤器，快速检测某一个 时间序列 是不是存在于文件内(这里等聊到 server 模块写文件的策略时候再聊)。我们知道这个过滤器的特点就是：没有的一定没有，但有的不一定有。为了保证准确性和过滤器序列化后的大小均衡，这里提供了一个 1% - 10% 错误率的可配置，当为 1% 错误率时，保存 1 万个测点信息，大概是 11.7 K。

我们再回想 SQL ： `SELECT 体温 FROM 王五 WHERE time = 1` 。读文件的过程就应该是：

1. 先用布隆过滤器判断文件内是否有王五的体温列，如果没有，查找下一个文件。
2. 从 DeviceMetaDataIndexMap 中找到王五的 TsDeviceMetaDataIndex ，从而得到了王五的 TsDeviceMetadata 的 offset，接下来就寻道至这个 offset 把王五的 TsDeviceMetadata 读出来。
3. MeasurementSchemaMap 不用关注，主要是给 Spark 使用的，ChunkHeader 中也保存了这些信息。

文件格式简介（三）(125)

评论排行榜

1. 时序数据库 Apache-IoTDB 源码解析之文件数据块（四）(3)
2. 时序数据库 Apache-IoTDB 源码解析之前言（一）(1)

TsDeviceMetaDataList

TsDeviceMetaDataList 属于文件的 2 级索引，用来索引具体的测点数据是不是存在、在哪里等信息。其中主要保存了：

1. ChunkGroupMetaData：ChunkGroup 的索引信息，主要包含了每个 ChunkGroup 数据块的起止位置以及包含的所有的测点元信息（ChunkMetaData）。
2. ChunkMetaData ： Chunk 的索引信息，主要包含了每个设备的测点在文件中的起止位置、开始结束时间、数据类型和预聚合信息。

上面的例子中，从 TsFileMetadata 已经拿到了王五的 TsDeviceMetadataIndex，这里就可以直接读出王五的 TsDeviceMetadata，并且遍历里边的 ChunkGroupMetadata 中的 ChunkMetadata，找到体温对应的所有的 ChunkMetadata。通过预聚合信息对时间过滤，判断能否使用当前的 Chunk 或者能否直接使用预聚合信息直接返回数据(等介绍到 server 的查询引擎时候细聊)。

如果不能直接返回，因为 ChunkMetaData 包含了这个 Chunk 对应的文件的偏移量，只需要使用 `seek(offSet)` 就会跳转到数据块，使用上一章介绍的读取方法进行遍历就完成了整个读取。

预聚合信息（Statistics）

文中多次提到了预聚合在这里详细介绍一下它的数据结构。

```
// 所属文件块的开始时间
private long startTime;
// 所属文件块的结束时间
private long endTime;
// 所属文件块的数据类型
```

```
private TSDataType tsDataType;
// 所属文件块的最小值
private int minValue;
// 所属文件块的最大值
private int maxValue;
// 所属文件块的第一个值
private int firstValue;
// 所属文件块的最后一个值
private int lastValue;
// 所属文件块的所有值的和
private double sumValue;
```

这个结构主要保存在 ChunkMetaData 和 PageHeader 中，这样做的好处就是，你不必从硬盘中读取具体的Page 或者 Chunk 的文件内容就可以获得最终的结果，例如：SELECT SUM(体温) FROM 王五 ，当定位到 ChunkMetaData 时，判断能否直接使用这个 Statistics 信息（具体怎么判断，之后会在介绍 server 时具体介绍），如果能使用，那么直接返回 sumValue。这样返回的速度，无论存了多少数据，它的聚合结果响应时间简直就是 1 毫秒以内。

样例数据

我们继续使用上一章聊到的示例数据来展示。

时间戳	人名	体温	心率
1580950800	王五	36.7	100
1580950911	王五	36.6	90

完整的文件信息如下：

```
             POSITION|   CONTENT
             --------    -------
                    0|   [magic head]
```

```
 TsFile
                    6|    [version number] 000002
                     // 数据块开始
||||||||||||||||||||    [Chunk Group] of wangwu begins at pos 12, ends at pos 253, version:0, num of Chunks:2
                   12|    [Chunk] of xinlv, numOfPoints:1, time range:[1580950800,1580950800], tsDataType:INT32,
                         [minValue:100,maxValue:100,firstValue:100,lastValue:100,sumValue:100.0]
                     |           [marker] 1
                     |           [ChunkHeader]
                     |           1 pages
                   121|    [Chunk] of tiwen, numOfPoints:1, time range:[1580950800,1580950800], tsDataType:FLOAT,
                         [minValue:36.7,maxValue:36.7,firstValue:36.7,lastValue:36.7,sumValue:36.7000007629345]
                     |           [marker] 1
                     |           [ChunkHeader]
                     |           1 pages
                   230|    [Chunk Group Footer]
```

```
                    |           [marker] 0
                    |           [deviceID] wangwu
                    |           [dataSize] 218
                    |           [num of chunks] 2
|||||||||||||||||||   [Chunk Group] of wangwu ends
                    // 索引块开始
                253|   [marker] 2
                254|   [TsDeviceMetadata] of wangwu, startTime:1580950800, endTime:1580950800
                    |           [startTime] 1580950800
                    |           [endTime] 1580950800
                    |           [ChunkGroupMetaData] of wangwu, startOffset12, endOffset253, version:0, numberOfChunks:2
                    |
[ChunkMetaData] of xinlv, startTime:1580950800, endTime:1580950800, offsetOfChunkHeader:12, dataType:INT32, statistics:[minValue:100,maxValue:100,firstValue:100,lastValue:100,sumValue:100.0]
                    |
[ChunkMetaData] of tiwen, startTime:1580950800, endTime:1580950800, offsetOfChunkHeader:121, dataType:FLOAT, statistics:[minValue:36.7,maxValue:36.7,firstValue:36.7,lastValue:36.7,sumValue:36.70000076293945]
```

```
                446|   [TsFileMetaData]
                   |           [num of devices] 1
                   |           [TsDeviceMetadataIndex] of wangwu, startTime:1580950800, endTime:1580950800, offSet:254, len:192
                   |           [num of measurements] 2
                   |           2 key&measurementSchema
                   |           [createBy isNotNull] false
                   |           [totalChunkNum] 2
                   |           [invalidChunkNum] 0
                   //布隆过滤器
                   |           [bloom filter bit vector byte array length] 30
                   |           [bloom filter bit vector byte array]
                   |           [bloom filter number of bits] 256
                   |           [bloom filter number of hash functions] 5
                599|   [TsFileMetaDataSize] 153
                603|   [magic tail] TsFile
                609|   END of TsFile
```

当执行： SELECT 体温 FROM 王五 时：

1. 从 599 开始读，1 级索引长度为 153.
2. 599 – 153 = 446 就是 1 级索引读开始位置，并读出 TsDeviceMetadataIndex of 王五，其中记录了，王五设备的 2 级索引的 offset 为 254.
3. 跳到 254 开始读 2 级索引，找到 ChunkMetaData of 体温， 其中记录了体温数据的 Chunk 的offset 为 121
4. 跳到 121 ，这里进入了数据块，从 121 读取到 230 ，读出的数据就全部是体温数据。

改进项

1. 只读投影列

前面第 3 步中，读取 2 级索引时候，会将这个设备下的所有测点数据全部读出来，这依然不太符合只读投影列的设计，所以在新的 TsFile 中，修改了 1级索引和 2 级索引的部分结构，使得读出的数据更少，更高效。有兴趣的同学可以关注 PR： Refactor TsFile #736

2. 文件级 Statistics

在物联网场景中经常会涉及到查询某个设备的最后状态，比如：车联网中，查询车辆的末次位置 (`SELECT LAST(lat,lon) FROM VechicleID`)，或者当前的点火、熄火状态等 `SELECT LAST(accStatus) FROM VechicleID` 。

或者当某些分页查询等情况时候，经常会使用到 `COUNT(*)` 等操作，这些都非常符合 Statistics 结构，这些场景涉及到的索引设计也都会体现到新的 TsFile 索引改动中。

到此已经介绍完了文件的整体结构，了解了大体的写入和读取过程，但是 TsFile 的 API 是如何设计的，怎样在代码里做一些特殊的功课，来绕过 Java 装箱、GC 等问题呢？欢迎持续关

注。。。。

标签: 列式数据库, 车联网, TsFile, IoTDB, 时序数据, 数据库, 数据库性能, 物联网, 行式数据库

Liutaohua
关注 - 0
粉丝 - 1
+加关注

« 上一篇： 时序数据库 Apache-IoTDB 源码解析之文件数据块（四）

posted @ 2020-02-14 14:55 Liutaohua 阅读(312) 评论(0) 编辑 收藏