UBC Android 时长不自洽修复方案 - 时长float精度不足

目录

时长不自洽问题整体现状

1. 背景&原因

`v13.58` 版本修复时长不自洽问题上线了 `endtime` 入库策略实验，以修复多进程场景下时长不自洽问题。在对 `5%` 流量实验组(***sid = 73780_2***)时长不自洽剩余日志(***pv = 12,344***)归因过程中发现：其中大部分日志(***pv = 11,099，占比89.9%***)是由于 `duration` 的精度不足导致的时长不自洽。

经线下自测，`endtime - starttime >= 16384001` 时，即实际时长大于约 `4.55` 小时，使用单精度浮点数 `float` 存储秒，小数点第三位数字就会受到精度不足的影响。从 `16384001` ~ `86400000` 即， `4.55` ~ `24` 小时，其中有 `54013572` 个毫秒数会受到精度问题影响，存在 `[-8,+8]` 范围的误差

原代码逻辑及示例

```java
public void flowSetValueWithDuration(Flow flow, String value) {
    // ...
```

```
    float diffStartTime = (float) (endTime - flow.getStartTime());                    // diffStartTime = 1.6384001E7 (16384001的科学计数法表示)
    float duration = diffStartTime / DateUtils.SECOND_IN_MILLIS;                      // duration = 16384.002
    if (duration < 0) {
        duration = 0;
    }
    object.put(Constants.UBC_DURATION, String.format(Locale.ENGLISH, "%.3f", duration));   // content$duration = '16384.002'; endtime - starttime = 16384001
    // ...
}
```

影响面：以5%流量实验组数据为例，duration精度不足导致的时长不自洽问题占大盘比例约0.030%(=11,099 / 36,216,210)；占实验组剩余不自洽日志比例约89.9%(=11,099 / 12,344)

收益：endtime入库策略实验实验组时长不自洽比例从 `0.034%` 降低到 `0.003%` $= 1,245/36,216,210$

2. 解决方案

方案1: 使用Double双精度浮点数

优点：只涉及数据类型的修改，修改风险低

缺点：理论上双精度浮点数仍有可能会有精度丢失问题（但考虑实际情况，双精度浮点数精度已经足够，时长（毫秒）在十万年的数量级精度仍足够使用）

</> SQL

```
public void flowSetValueWithDuration(Flow flow, String value) {
    // ...
```

```
    long diffStartTime = endTime - flow.getStartTime();
    if (diffStartTime < 0) {
        diffStartTime = 0;
    }
    object.put(Constants.UBC_DURATION, String.format(Locale.ENGLISH, "%.3f", (double) diffStartTime / DateUtils.SECOND_IN_MILLIS));
    // ...
}
```

方案2: 使用BigDecimal

优点：彻底解决精度问题

缺点：计算上的内存和性能损耗更大；使用BigDecimal的API处理时间和转字符串逻辑，API使用不当可能会引入风险

</> SQL

```
public void flowSetValueWithDuration(Flow flow, String value) {
    // ...
    long diffStartTime = endTime - flow.getStartTime();
    if (diffStartTime < 0) {
        diffStartTime = 0;
    }
    object.put(Constants.UBC_DURATION, BigDecimal.valueOf(diffStartTime).divide(BigDecimal.valueOf(DateUtils.SECOND_IN_MILLIS), 3, RoundingMode.HALF_UP).toPlainString());
    // ...
}

```

附录

Decimal 和 String.format 性能比较

线下多次测试 `Decimal` 和 `String.format()` 没有明显优劣，平均单次调用都在 `500ns` 左右