在项目中，随着时间的推移，数据量越来越大，程序的某些功能性能也可能会随之下降，那么此时我们不得不需要对之前的功能进行性能优化。如果优化方案不得当，或者说不优雅，那可能将对整个系统产生不可逆的严重影响，以下是对大数据分布式项目的SQL优化方案。

我们常见的SQL优化方案有，1.实时sql优化，2.业务冗余字段，3.离线计算

1.实时sql优化：就是将分析出来耗时的sql进行重写、拆分成多次查询后数据重组、去掉sql函数等等；sql能干的事情，程序肯定能干，且程序运行的性能一般情况会快很多，而且web服务器可以部署很多台；优点：可实现快速优化，且性能非常可观；缺点：可能会增加程序的复杂度；

2.业务冗余字段：就是在更新某张数据库业务表时，将其关联表的某些字段冗余进来，以减少某些业务的表关联查询从而达到提升速度效果；优点：实现简单；缺点：此方案只适用于某些比较简单的场景，如果关联的表非常多，且业务需要查询的字段也非常多，此时程序将对其它业务表的业务产生非常严重的侵入【即在更新别的表时候，需要更新该表】，在后期优化过程中，该方案还涉及到初始化数据的问题，另外数据一致性问题堪忧【如多业务保存中途宕机】，所以复杂业务不建议使用该方案。

3.离线计算：就是利用定时任务或者hadoop等到一些离线计算的技术，将数据跑成报表的方式。此方案适用于可支持非实时数据的查看的业务【如报表等等】。优点，可单独部署，不影响主程序，且性能优化效果非常可靠；缺点：数据非实时，且可能出现报表数据和真是数据某些字段不一致的情况，程序复杂度倍增且需要额外服务器支撑。

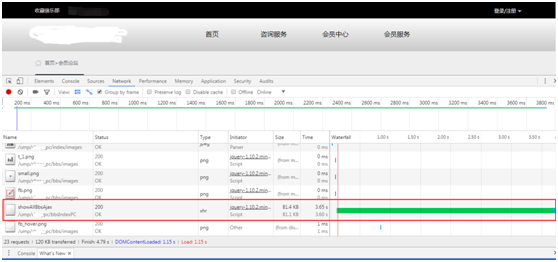
在优化方案选择的优先级别上：

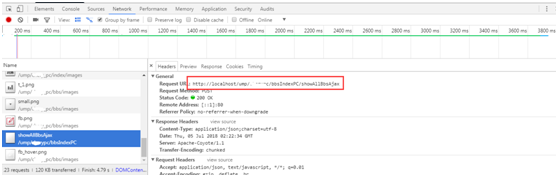
实时sql优化>离线计算>数据冗余；

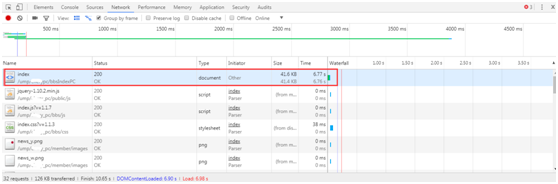
注意：在某些特定场景下【如业务非常简单时】，离线计算的优先级别小于数据冗余。

定位步骤[本次以后台sql导致的性能为例，其它如前端或程序处理导致性能问题不考虑]:

1.使用google浏览器按F12，打开性能差页面，检查耗时的请求







2.根据请求找到访问的controller-service-dbs-->sql

3.sql性能分析【将sql用生产数据库进行运行分析，此次不使用explain分析的方法】

场景一：一般都是复杂SQL查询导致的慢查询，我们可以将出现慢查询的SQL，提取出来，使用线上的数据库进行执行（比如使用navicat，SQLyog)，可以明显的看出SQL的执行速度。

对于复杂SQL的方案基本上都是将SQL拆解为多个简单SQL。

场景二：需要分页的数据来自多张表(一般是表结构相同，数据含义不同)，原SQL如下。

|  |
| --- |
| SELECT \* FROM  (  SELECT \* FROM t\_a a WHERE a.time > '2018-07-01'  UNION ALL  SELECT \* FROM t\_b b WHERE b.time > '2018-07-01'  UNION ALL  SELECT \* FROM t\_c c WHERE c.time > '2018-07-01'  ) tp  ORDER BY tp.time DESC  LIMIT 0, 10 |

上面这条sql中包含了两个union all,也句是由3条sql结果合并组成，这里是有排序的，且为分页查询，问题可不是分sql查询这么简单。所以我们采用的是SQL经典优化案例，sql性能衰减方案，该方案就是先根据当前页\*每页大小（该值名称定义为pageTotal，以下将以此名称表示），然后将多个子句按照上诉得出来的pageTotal值查询出pageTotal条数据，然后进行排序取出当前查询的那页的数据。

|  |
| --- |
| SELECT \* FROM t\_a WHERE TIME > '2018-07-01' ORDER BY TIME DESC LIMIT 0, 10;  SELECT \* FROM t\_b WHERE TIME > '2018-07-01' ORDER BY TIME DESC LIMIT 0, 10;  SELECT \* FROM t\_c WHERE TIME > '2018-07-01' ORDER BY TIME DESC LIMIT 0, 10; |

将一个大的SQL分为三个小SQL，每一个SQL都返回pageTotal相同的数据，然后我们将返回的数据在程序中进行排序筛选。

但随着页数的增大，limit 后面的10就成当前页的大小的倍数增大，所以当前页越大时，需要查询运算的数据就会越来越多，性能就会进行衰减，且此方法会将数据load到内存占用内存空间，使用者需权衡使用。一个很好的解决到了一定页面后，性能衰减到客户不能承受的时间差时，我们可以判断页数到达多少页时，使用原来一条sql的查询，保证功能可使用。此方案适用于列表在业务上查询使用新数据多，老数据几乎不用的场景。

场景三：多个表的left join 查询。页面中的查询条件多而复杂，下面的SQL可能完全不可用。

页面查询表单分析，子句中只有一个vin码是查询条件，vin对应唯一的一辆车，一辆车又对应唯一的一个车主会员，于是我们可以将所有子句拆分出来分批查询组装后返回；思路如下，在没有vin查询时候，我们根据上面条件单表查询wcc\_ch\_member得出当前页的10条数据的id集合，然后根据这10条数据使用memberid in（......）的方式分批次去查询后面的统计字段和其他子句的字段进行组装。经分析，所有子句查询都在0.01秒左右，且wcc\_ch\_member在单表查询是也是0.02秒左右，由此可推算出第一种情况查询优化效果提升千万倍性能。那么在有vin码条件的情况下呢，我们首先可以根据vin码查询出member表的会员id,然后根据id再去按照上面的方法查询对应的统计，分析出来只多了一步member查询，性能上和上诉无vin码查询是效果几乎无变化，且可能更快。

SELECT
m.id,
m.member\_name,
m.phone\_no,
CASE
WHEN m.sex = '1' THEN
'男'
WHEN m.sex = '2' THEN
'女'
END sex,
m.email,
CASE
WHEN m.member\_level = '0' THEN
'游客'
WHEN m.member\_level = '1' THEN
'普卡'
WHEN m.member\_level = '2' THEN
'银卡'
WHEN m.member\_level = '3' THEN
'金卡'
END member\_level,
m.registration\_time,
m.expiration\_time,
s. NAME,
d.dealer\_name,
d.dealer\_no,
n.vin\_code,
n.carType,
j.canUseScore,
j1.usedScore,
j2.sumScore
FROM
wcc\_ch\_member m
LEFT JOIN (
SELECT
v.wcc\_member\_info,
GROUP\_CONCAT(
t.vehicle\_type\_name SEPARATOR ';'
) AS carType,
GROUP\_CONCAT("'", v.vin\_code, "'") AS vin\_code
FROM
wcc\_ch\_vehicle v,
wcc\_vehicle\_type t
WHERE
v.wcc\_vehicle\_type = t.id
AND (v. STATUS = 1 OR v. STATUS = 3)
GROUP BY
v.wcc\_member\_info
) n ON n.wcc\_member\_info = m.id
LEFT JOIN wcc\_friend h ON h.open\_id = m.open\_id
LEFT JOIN wcc\_sale\_assist s ON h.said = s.id
LEFT JOIN wcc\_ch\_dealer d ON s.dealer = d.id
LEFT JOIN (
SELECT
wcc\_member\_info,
sum(can\_use\_score) canUseScore
FROM
wcc\_ch\_integral\_detail
WHERE
isadd = 1
AND can\_use\_score > 0
AND (
expiration\_Date IS NULL
OR expiration\_date >= '2018-07-05 14:00:34'
)
GROUP BY
wcc\_member\_info
) j ON m.id = j.wcc\_member\_info
LEFT JOIN (
SELECT
wcc\_member\_info,
sum(score - return\_score) usedScore
FROM
wcc\_ch\_integral\_detail
WHERE
isadd = '0'
GROUP BY
wcc\_member\_info
) j1 ON m.id = j1.wcc\_member\_info
LEFT JOIN (
SELECT
wcc\_member\_info,
sum(score) sumScore
FROM
wcc\_ch\_integral\_detail
WHERE
isadd = '1'
GROUP BY
wcc\_member\_info
) j2 ON m.id = j2.wcc\_member\_info
WHERE
1 = 1
LIMIT 0,
10

总结一下，所有子句的查询均可以分批次查询来优化，所有left join类型的查询其实均可转为单表查询