

# 整理版日志

- 日志内容：
- 详细分析：
  - 查询语句推荐
  - 关键词识别映射
  - 生成初步S2SQL并修正（开始大模型生成过程）
    - JSON展开解释：
    - SQL转化展开解释：
    - 日志展开解释：
  - Apache Calcite 原理解析：
    - 主要功能：
    - 其他资料：
    - Calcite相关日志解析：
  - 查询优化（Apache Calcite框架）
  - 第二轮查询优化
  - 第三轮查询优化
  - 开始查询
  - 反复出现的数据库查询操作示例（包含查询指标、聊天记录等）：

## 日志内容：

排除了所有SQL库查询操作日志

```
14:56:57 [http-nio-9080-exec-10] DEBUG o.s.web.servlet.Dispatche
14:56:57 [http-nio-9080-exec-10] DEBUG o.s.w.s.m.m.a.RequestMapp
14:56:57 [http-nio-9080-exec-10] DEBUG o.s.w.s.m.m.a.RequestResp
14:56:57 [ForkJoinPool.commonPool-worker-19] DEBUG c.h.h.c.trie.
14:56:57 [ForkJoinPool.commonPool-worker-19] DEBUG c.h.h.c.trie.
14:56:57 [http-nio-9080-exec-10] DEBUG c.t.s.h.s.f.s.i.RetrieveS
14:56:57 [http-nio-9080-exec-10] DEBUG c.t.s.h.s.f.s.i.RetrieveS
14:56:57 [http-nio-9080-exec-10] DEBUG c.t.s.h.s.f.s.i.RetrieveS
14:56:57 [http-nio-9080-exec-10] DEBUG c.t.s.h.s.f.s.i.RetrieveS
14:56:57 [http-nio-9080-exec-10] INFO c.t.s.h.s.f.s.i.RetrieveS
14:56:57 [http-nio-9080-exec-10] DEBUG c.t.s.h.s.f.s.i.RetrieveS
14:56:57 [http-nio-9080-exec-10] DEBUG o.s.w.s.m.m.a.RequestResp
14:56:57 [http-nio-9080-exec-10] DEBUG o.s.w.s.m.m.a.RequestResp
14:56:59 [http-nio-9080-exec-3] DEBUG o.s.web.servlet.Dispatcher
14:56:59 [http-nio-9080-exec-3] DEBUG o.s.w.s.m.m.a.RequestMappi
14:56:59 [http-nio-9080-exec-3] DEBUG o.s.w.s.m.m.a.RequestRespo
```

```
14:56:59 [http-nio-9080-exec-3] DEBUG o.s.web.servlet.Dispatcher
14:56:59 [http-nio-9080-exec-3] DEBUG o.s.w.s.m.m.a.RequestMappi
14:56:59 [http-nio-9080-exec-3] DEBUG o.s.w.s.m.m.a.RequestRespo
14:56:59 [ForkJoinPool.commonPool-worker-12] DEBUG c.h.h.c.trie.
14:56:59 [ForkJoinPool.commonPool-worker-12] DEBUG c.h.h.c.trie.
14:56:59 [ForkJoinPool.commonPool-worker-12] DEBUG c.h.h.c.trie.
14:56:59 [ForkJoinPool.commonPool-worker-12] DEBUG c.h.h.c.trie.
14:56:59 [ForkJoinPool.commonPool-worker-12] DEBUG c.h.h.c.trie.
14:56:59 [ForkJoinPool.commonPool-worker-12] DEBUG c.h.h.c.trie.
14:56:59 [ForkJoinPool.commonPool-worker-12] DEBUG c.h.h.c.trie.
14:56:59 [ForkJoinPool.commonPool-worker-12] DEBUG c.h.h.c.trie.
14:56:59 [ForkJoinPool.commonPool-worker-12] DEBUG c.h.h.c.trie.
14:56:59 [http-nio-9080-exec-3] DEBUG c.t.s.h.s.f.s.i.RetrieveSe
14:56:59 [http-nio-9080-exec-3] DEBUG c.t.s.h.s.f.s.i.RetrieveSe
14:56:59 [http-nio-9080-exec-3] DEBUG c.t.s.h.s.f.s.i.RetrieveSe
14:56:59 [http-nio-9080-exec-3] DEBUG c.t.s.h.s.f.s.i.RetrieveSe
14:56:59 [http-nio-9080-exec-3] DEBUG c.t.s.h.s.f.s.i.RetrieveSe
14:56:59 [http-nio-9080-exec-3] DEBUG c.t.s.h.s.f.s.i.RetrieveSe
14:56:59 [http-nio-9080-exec-3] INFO c.t.s.h.s.f.s.i.RetrieveSe
14:56:59 [http-nio-9080-exec-3] DEBUG c.t.s.h.s.f.s.i.RetrieveSe
14:56:59 [http-nio-9080-exec-3] DEBUG o.s.w.s.m.m.a.RequestRespo
14:56:59 [http-nio-9080-exec-3] DEBUG o.s.w.s.m.m.a.RequestRespo
14:56:59 [http-nio-9080-exec-3] DEBUG o.s.web.servlet.Dispatcher
14:57:00 [scheduling-1] DEBUG c.t.s.h.s.task.DictionaryReloadTas
14:57:00 [scheduling-1] DEBUG c.t.s.h.s.w.s.impl.DictWordService
14:57:00 [scheduling-1] DEBUG c.t.s.h.s.w.s.impl.DictWordService
14:57:00 [scheduling-1] DEBUG c.t.s.h.s.w.s.impl.DictWordService
14:57:00 [scheduling-1] DEBUG c.t.s.h.s.w.s.impl.DictWordService
14:57:00 [scheduling-1] DEBUG c.t.s.h.s.w.s.impl.DictWordService
14:57:00 [scheduling-1] DEBUG c.t.s.h.s.w.s.impl.DictWordService
14:57:00 [scheduling-1] DEBUG c.t.s.h.s.task.DictionaryReloadTas
14:57:02 [http-nio-9080-exec-8] DEBUG o.s.web.servlet.Dispatcher
14:57:02 [http-nio-9080-exec-8] DEBUG o.s.w.s.m.m.a.RequestMappi
14:57:02 [http-nio-9080-exec-8] DEBUG o.s.w.s.m.m.a.RequestRespo
14:57:02 [ForkJoinPool.commonPool-worker-5] DEBUG c.h.h.c.trie.b
14:57:02 [ForkJoinPool.commonPool-worker-5] DEBUG c.h.h.c.trie.b
14:57:02 [ForkJoinPool.commonPool-worker-5] DEBUG c.h.h.c.trie.b
14:57:02 [ForkJoinPool.commonPool-worker-5] DEBUG c.h.h.c.trie.b
14:57:02 [ForkJoinPool.commonPool-worker-5] DEBUG c.h.h.c.trie.b
14:57:02 [ForkJoinPool.commonPool-worker-5] DEBUG c.h.h.c.trie.b
```

```
14:57:02 [http-nio-9080-exec-8] DEBUG c.t.s.h.s.f.s.i.RetrieveSe
14:57:02 [http-nio-9080-exec-8] DEBUG c.t.s.h.s.f.s.i.RetrieveSe
14:57:02 [http-nio-9080-exec-8] DEBUG c.t.s.h.s.f.s.i.RetrieveSe
14:57:02 [http-nio-9080-exec-8] DEBUG c.t.s.h.s.f.s.i.RetrieveSe
14:57:02 [http-nio-9080-exec-8] DEBUG c.t.s.h.s.f.s.i.RetrieveSe
14:57:02 [http-nio-9080-exec-8] DEBUG c.t.s.h.s.f.s.i.RetrieveSe
14:57:02 [http-nio-9080-exec-8] INFO c.t.s.h.s.f.s.i.RetrieveSe
14:57:02 [http-nio-9080-exec-8] DEBUG c.t.s.h.s.f.s.i.RetrieveSe
14:57:02 [http-nio-9080-exec-8] DEBUG o.s.w.s.m.m.a.RequestRespo
14:57:02 [http-nio-9080-exec-8] DEBUG o.s.w.s.m.m.a.RequestRespo
14:57:02 [http-nio-9080-exec-8] DEBUG o.s.web.servlet.Dispatcher
14:57:04 [http-nio-9080-exec-2] DEBUG o.s.web.servlet.Dispatcher
14:57:04 [http-nio-9080-exec-2] DEBUG o.s.w.s.m.m.a.RequestMappi
14:57:04 [http-nio-9080-exec-2] DEBUG o.s.w.s.m.m.a.RequestRespo
14:57:04 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.mapper.BaseMatch
14:57:04 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.mapper.BaseMatch
14:57:06 [http-nio-9080-exec-2] DEBUG c.t.s.h.chat.mapper.BaseMa
14:57:06 [http-nio-9080-exec-2] DEBUG c.t.s.h.chat.mapper.BaseMa
14:57:06 [http-nio-9080-exec-2] DEBUG c.t.s.h.chat.mapper.BaseMa
14:57:06 [http-nio-9080-exec-2] DEBUG c.t.s.h.chat.mapper.BaseMa
14:57:06 [http-nio-9080-exec-2] DEBUG c.t.s.h.s.w.s.i.WorkflowSe
14:57:07 [http-nio-9080-exec-2] INFO c.t.s.h.chat.parser.llm.LL
14:57:19 [http-nio-9080-exec-2] DEBUG c.t.s.h.s.w.s.i.WorkflowSe
14:57:19 [http-nio-9080-exec-2] DEBUG c.t.s.h.s.w.s.i.WorkflowSe
14:57:19 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.c.BaseSemanticCo
14:57:19 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.c.BaseSemanticCo
14:57:19 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.c.BaseSemanticCo
14:57:19 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.c.BaseSemanticCo
14:57:19 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.c.BaseSemanticCo
14:57:19 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.c.BaseSemanticCo
14:57:19 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.c.BaseSemanticCo
14:57:19 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.c.BaseSemanticCo
14:57:19 [http-nio-9080-exec-2] DEBUG c.t.s.h.s.w.s.impl.SchemaS
14:57:19 [http-nio-9080-exec-2] DEBUG c.t.s.h.s.w.s.impl.SchemaS
14:57:20 [http-nio-9080-exec-2] DEBUG c.t.s.h.s.utils.QueryReqCo
14:57:20 [http-nio-9080-exec-2] DEBUG c.t.s.h.s.utils.QueryReqCo
14:57:20 [http-nio-9080-exec-2] DEBUG c.t.s.h.s.utils.QueryReqCo
14:57:20 [http-nio-9080-exec-2] DEBUG c.t.s.h.s.utils.QueryReqCo
```

```

14:57:20 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.u.SysTimeDimensi
14:57:20 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.u.SysTimeDimensi
14:57:20 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.u.SysTimeDimensi
14:57:20 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.u.SysTimeDimensi
14:57:20 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.u.SysTimeDimensi
14:57:20 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.t.DefaultSemanti
14:57:20 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.t.DefaultSemanti
14:57:20 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.t.DefaultSemanti
14:57:20 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.t.DefaultSemanti
14:57:20 [http-nio-9080-exec-2] INFO c.t.s.h.c.t.DefaultSemanti
14:57:20 [http-nio-9080-exec-2] INFO c.t.s.h.c.t.c.s.node.DataS
14:57:20 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.t.c.s.node.DataS
14:57:20 [http-nio-9080-exec-2] DEBUG org.apache.calcite.sql.par
14:57:22 [http-nio-9080-exec-2] DEBUG org.apache.calcite.sql2rel
LogicalProject(sys_imp_date=[0], pv=[2])
  LogicalAggregate(group=[{0, 1}], s2_pv_uv_statis_pv=[SUM(2)])
    LogicalProject(sys_imp_date=[4], imp_date=[4], s2_pv_uv_st
      LogicalTableScan(table=[[s2_pv_uv_statis]])

14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.t.c.sql.node.Sem
FROM
s2_pv_uv_statis
GROUP BY `imp_date`, `imp_date`
14:57:22 [http-nio-9080-exec-2] DEBUG o.a.c.p.A.rule_execution_s
14:57:22 [http-nio-9080-exec-2] DEBUG o.a.c.p.A.rule_execution_s
Rules
* Total

14:57:22 [http-nio-9080-exec-2] DEBUG o.apache.calcite.plan.RelO
14:57:22 [http-nio-9080-exec-2] DEBUG o.apache.calcite.plan.RelO
14:57:22 [http-nio-9080-exec-2] DEBUG o.apache.calcite.plan.RelO
14:57:22 [http-nio-9080-exec-2] DEBUG o.apache.calcite.plan.RelO
14:57:22 [http-nio-9080-exec-2] DEBUG org.apache.calcite.sql.par
14:57:22 [http-nio-9080-exec-2] DEBUG org.apache.calcite.sql2rel
LogicalAggregate(group=[{}], EXPR$0=[SUM(0)])
  LogicalProject(pv=[1])
    LogicalFilter(condition=[AND(>=(0, _UTF-8'2024-07-31'), <=(
      LogicalProject(sys_imp_date=[0], pv=[1])
        LogicalAggregate(group=[{0}], pv=[SUM(1)])

```

```

        LogicalProject(imp_date=[$4], $f1=[1])
        LogicalTableScan(table=[[s2_pv_uv_statis]])

14:57:22 [http-nio-9080-exec-2] DEBUG org.apache.calcite.sql2rel
LogicalAggregate(group=[{}], EXPR$0=[SUM($0)])
    LogicalProject(pv=[$1])
        LogicalFilter(condition=[AND(>=($0, _UTF-8'2024-07-31'), <=(
            LogicalProject(sys_imp_date=[$0], pv=[$1])
            LogicalAggregate(group=[{0}], pv=[SUM($1)])
            LogicalProject(imp_date=[$4], $f1=[1])
            LogicalTableScan(table=[[s2_pv_uv_statis]])

14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.t.c.sql.node.Sem
FROM
(SELECT `imp_date` AS `sys_imp_date`, SUM(1) AS `pv`
FROM
s2_pv_uv_statis
GROUP BY `imp_date`) AS `t1`
WHERE `sys_imp_date` >= '2024-07-31' AND `sys_imp_date` <= '2024
14:57:22 [http-nio-9080-exec-2] DEBUG o.apache.calcite.plan.RelO
14:57:22 [http-nio-9080-exec-2] DEBUG o.a.c.p.A.rule_execution_s
14:57:22 [http-nio-9080-exec-2] DEBUG o.a.c.p.A.rule_execution_s
Rules
FilterToGroupScanRule
* Total

14:57:22 [http-nio-9080-exec-2] DEBUG o.apache.calcite.plan.RelO
14:57:22 [http-nio-9080-exec-2] DEBUG o.apache.calcite.plan.RelO
14:57:22 [http-nio-9080-exec-2] DEBUG o.apache.calcite.plan.RelO
14:57:22 [http-nio-9080-exec-2] DEBUG o.apache.calcite.plan.RelO
14:57:22 [http-nio-9080-exec-2] DEBUG o.apache.calcite.plan.RelO
14:57:22 [http-nio-9080-exec-2] DEBUG o.apache.calcite.plan.RelO
14:57:22 [http-nio-9080-exec-2] DEBUG o.apache.calcite.plan.RelO
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.t.c.CalciteQuery
FROM
(SELECT `imp_date` AS `sys_imp_date`, SUM(1) AS `pv`
FROM
s2_pv_uv_statis
GROUP BY `imp_date`) AS `t7`
WHERE `sys_imp_date` >= '2024-07-31' AND `sys_imp_date` <= '2024

```

```

14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.t.DetailQueryOpt
FROM
s2_pv_uv_statistic
GROUP BY `imp_date`, `imp_date`)
SELECT SUM(pv) FROM t_1 WHERE (sys_imp_date >= '2024-07-31' AND
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.t.DetailQueryOpt
FROM
s2_pv_uv_statistic
GROUP BY `imp_date`, `imp_date`)
SELECT SUM(pv) FROM t_1 WHERE (sys_imp_date >= '2024-07-31' AND
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.s.utils.QueryReqCo
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.s.utils.QueryReqCo
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.s.utils.QueryReqCo
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.s.utils.QueryReqCo
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.s.utils.QueryReqCo
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.u.SysTimeDimensi
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.u.SysTimeDimensi
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.u.SysTimeDimensi
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.u.SysTimeDimensi
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.t.DefaultSemanti
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.t.DefaultSemanti
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.t.DefaultSemanti
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.t.DefaultSemanti
14:57:22 [http-nio-9080-exec-2] INFO c.t.s.h.c.t.DefaultSemanti
14:57:22 [http-nio-9080-exec-2] INFO c.t.s.h.c.t.DefaultSemanti
14:57:22 [http-nio-9080-exec-2] INFO c.t.s.h.c.t.c.s.node.DataS
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.t.c.s.node.DataS
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.t.c.s.node.DataS
14:57:22 [http-nio-9080-exec-2] INFO c.t.s.h.c.t.c.sql.render.J
14:57:22 [http-nio-9080-exec-2] DEBUG org.apache.calcite.sql.par
14:57:22 [http-nio-9080-exec-2] DEBUG org.apache.calcite.sql.par
14:57:22 [http-nio-9080-exec-2] INFO c.t.s.h.c.t.c.sql.render.J
14:57:22 [http-nio-9080-exec-2] DEBUG org.apache.calcite.sql.par
14:57:22 [http-nio-9080-exec-2] DEBUG org.apache.calcite.sql2rel
LogicalProject(sys_imp_date=[3], department=[1], pv=[2])
  LogicalJoin(condition=[($0, $4)], joinType=[left])
    LogicalProject(user_name=[0], department=[1])
      LogicalTableScan(table=[[s2_user_department]])
        LogicalProject(s2_pv_uv_statistic_pv=[1], sys_imp_date=[4], us

```



```

        LogicalTableScan(table=[[s2_pv_uv_statistic]])

14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.t.c.sql.node.Sem
FROM
(SELECT `user_name`, `department`
FROM
s2_user_department) AS `t`
LEFT JOIN (SELECT 1 AS `s2_pv_uv_statistic_pv`, `imp_date` AS `sys_
FROM
s2_pv_uv_statistic) AS `t0` ON `t`.`user_name` = `t0`.`user_name`
14:57:22 [http-nio-9080-exec-2] DEBUG o.a.c.p.A.rule_execution_s
14:57:22 [http-nio-9080-exec-2] DEBUG o.a.c.p.A.rule_execution_s
Rules
* Total

14:57:22 [http-nio-9080-exec-2] DEBUG o.apache.calcite.plan.RelO
14:57:22 [http-nio-9080-exec-2] DEBUG o.apache.calcite.plan.RelO
14:57:22 [http-nio-9080-exec-2] DEBUG o.apache.calcite.plan.RelO
14:57:22 [http-nio-9080-exec-2] DEBUG o.apache.calcite.plan.RelO
14:57:22 [http-nio-9080-exec-2] DEBUG o.apache.calcite.plan.RelO
14:57:22 [http-nio-9080-exec-2] DEBUG o.apache.calcite.plan.RelO
14:57:22 [http-nio-9080-exec-2] DEBUG org.apache.calcite.sql.par
14:57:22 [http-nio-9080-exec-2] DEBUG org.apache.calcite.sql.par
14:57:22 [http-nio-9080-exec-2] DEBUG org.apache.calcite.sql2rel
LogicalSort(fetch=[365])
    LogicalAggregate(group=[{0, 1}], pv=[SUM($2)])
        LogicalFilter(condition=[AND(>=($0, _UTF-8'2024-07-08'), <=
            LogicalProject(sys_imp_date=[$3], department=[$1], pv=[$2]
                LogicalJoin(condition=[=($0, $4)], joinType=[left])
                    LogicalProject(user_name=[$0], department=[$1])
                        LogicalTableScan(table=[[s2_user_department]])
                            LogicalProject(s2_pv_uv_statistic_pv=[1], sys_imp_date=[$
                                LogicalTableScan(table=[[s2_pv_uv_statistic]])

14:57:22 [http-nio-9080-exec-2] DEBUG org.apache.calcite.sql2rel
LogicalSort(fetch=[365])
    LogicalAggregate(group=[{0, 1}], pv=[SUM($2)])
        LogicalFilter(condition=[AND(>=($0, _UTF-8'2024-07-08'), <=
            LogicalProject(sys_imp_date=[$3], department=[$1], pv=[$2]
                LogicalJoin(condition=[=($0, $4)], joinType=[left])

```

```

        LogicalProject(user_name=[$0], department=[$1])
        LogicalTableScan(table=[[s2_user_department]])
        LogicalProject(s2_pv_uv_statis_pv=[1], sys_imp_date=[$
        LogicalTableScan(table=[[s2_pv_uv_statis]])

14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.t.c.sql.node.Sem
FROM
(SELECT `t0`.`sys_imp_date`, `t`.`department`, `t0`.`s2_pv_uv_st
FROM
(SELECT `user_name`, `department`
FROM
s2_user_department) AS `t`
LEFT JOIN (SELECT 1 AS `s2_pv_uv_statis_pv`, `imp_date` AS `sys_
FROM
s2_pv_uv_statis) AS `t0` ON `t`.`user_name` = `t0`.`user_name`)
WHERE `t1`.`sys_imp_date` >= '2024-07-08' AND `t1`.`sys_imp_date`
GROUP BY `sys_imp_date`, `department`
LIMIT 365
14:57:22 [http-nio-9080-exec-2] DEBUG o.a.c.p.A.rule_execution_s
14:57:22 [http-nio-9080-exec-2] DEBUG o.a.c.p.A.rule_execution_s
Rules
* Total

14:57:22 [http-nio-9080-exec-2] DEBUG o.apache.calcite.plan.RelO
14:57:22 [http-nio-9080-exec-2] DEBUG o.apache.calcite.plan.RelO
14:57:22 [http-nio-9080-exec-2] DEBUG o.apache.calcite.plan.RelO
14:57:22 [http-nio-9080-exec-2] DEBUG o.apache.calcite.plan.RelO
14:57:22 [http-nio-9080-exec-2] DEBUG o.apache.calcite.plan.RelO
14:57:22 [http-nio-9080-exec-2] DEBUG o.apache.calcite.plan.RelO
14:57:22 [http-nio-9080-exec-2] DEBUG o.apache.calcite.plan.RelO
14:57:22 [http-nio-9080-exec-2] DEBUG o.apache.calcite.plan.RelO
14:57:22 [http-nio-9080-exec-2] DEBUG o.apache.calcite.plan.RelO
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.t.c.CalciteQuery
FROM
(SELECT `t6`.`sys_imp_date`, `t5`.`department`, `t6`.`s2_pv_uv_s
FROM
(SELECT `user_name`, `department`
FROM
s2_user_department) AS `t5`
LEFT JOIN (SELECT 1 AS `s2_pv_uv_statis_pv`, `imp_date` AS `sys_

```



```

FROM
s2_pv_uv_statistic) AS `t6` ON `t5`.`user_name` = `t6`.`user_name`)
WHERE `t7`.`sys_imp_date` >= '2024-07-08' AND `t7`.`sys_imp_date`
GROUP BY `sys_imp_date`, `department`
LIMIT 365]
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.t.DetailQueryOpt
FROM
(SELECT `user_name`, `department`
FROM
s2_user_department) AS `t2`
LEFT JOIN (SELECT 1 AS `s2_pv_uv_statistic_pv`, `imp_date` AS `sys_
FROM
s2_pv_uv_statistic) AS `t3` ON `t2`.`user_name` = `t3`.`user_name`)
SELECT sys_imp_date, department, SUM(pv) AS pv FROM t_1 WHERE sy
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.t.DetailQueryOpt
FROM
(SELECT `user_name`, `department`
FROM
s2_user_department) AS `t2`
LEFT JOIN (SELECT 1 AS `s2_pv_uv_statistic_pv`, `imp_date` AS `sys_
FROM
s2_pv_uv_statistic) AS `t3` ON `t2`.`user_name` = `t3`.`user_name`)
SELECT sys_imp_date, department, SUM(pv) AS pv FROM t_1 WHERE sy
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.s.utils.QueryReqCo
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.s.utils.QueryReqCo
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.s.utils.QueryReqCo
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.s.utils.QueryReqCo
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.s.utils.QueryReqCo
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.u.SysTimeDimensi
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.u.SysTimeDimensi
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.u.SysTimeDimensi
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.u.SysTimeDimensi
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.u.SysTimeDimensi
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.t.DefaultSemanti
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.t.DefaultSemanti
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.t.DefaultSemanti
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.t.DefaultSemanti
14:57:22 [http-nio-9080-exec-2] INFO c.t.s.h.c.t.DefaultSemanti
14:57:22 [http-nio-9080-exec-2] INFO c.t.s.h.c.t.DefaultSemanti
14:57:22 [http-nio-9080-exec-2] INFO c.t.s.h.c.t.c.s.node.DataS

```

```

14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.t.c.s.node.DataS
14:57:22 [http-nio-9080-exec-2] DEBUG org.apache.calcite.sql.par
14:57:22 [http-nio-9080-exec-2] DEBUG org.apache.calcite.sql.par
14:57:22 [http-nio-9080-exec-2] DEBUG org.apache.calcite.sql.par
14:57:22 [http-nio-9080-exec-2] DEBUG org.apache.calcite.sql2rel
LogicalProject(sys_imp_date=[$4], pv=[1])
  LogicalTableScan(table=[[s2_pv_uv_statistic]])

14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.t.c.sql.node.Sem
FROM
s2_pv_uv_statistic
14:57:22 [http-nio-9080-exec-2] DEBUG o.a.c.p.A.rule_execution_s
14:57:22 [http-nio-9080-exec-2] DEBUG o.a.c.p.A.rule_execution_s
Rules
* Total

14:57:22 [http-nio-9080-exec-2] DEBUG o.apache.calcite.plan.RelO
14:57:22 [http-nio-9080-exec-2] DEBUG o.apache.calcite.plan.RelO
14:57:22 [http-nio-9080-exec-2] DEBUG org.apache.calcite.sql.par
14:57:22 [http-nio-9080-exec-2] DEBUG org.apache.calcite.sql2rel
LogicalSort(fetch=[365])
  LogicalAggregate(group=[{0}], pv=[SUM($1)])
    LogicalFilter(condition=[AND(>=($0, _UTF-8'2024-07-08'), <=
      LogicalProject(sys_imp_date=[$4], pv=[1])
        LogicalTableScan(table=[[s2_pv_uv_statistic]])

14:57:22 [http-nio-9080-exec-2] DEBUG org.apache.calcite.sql2rel
LogicalSort(fetch=[365])
  LogicalAggregate(group=[{0}], pv=[SUM($1)])
    LogicalFilter(condition=[AND(>=($0, _UTF-8'2024-07-08'), <=
      LogicalProject(sys_imp_date=[$4], pv=[1])
        LogicalTableScan(table=[[s2_pv_uv_statistic]])

14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.t.c.sql.node.Sem
FROM
(SELECT `imp_date` AS `sys_imp_date`, 1 AS `pv`
FROM
s2_pv_uv_statistic) AS `t`
WHERE `sys_imp_date` >= '2024-07-08' AND `sys_imp_date` <= '2024
GROUP BY `sys_imp_date`

```

```

LIMIT 365
14:57:22 [http-nio-9080-exec-2] DEBUG o.a.c.p.A.rule_execution_s
14:57:22 [http-nio-9080-exec-2] DEBUG o.a.c.p.A.rule_execution_s
Rules
* Total

14:57:22 [http-nio-9080-exec-2] DEBUG o.apache.calcite.plan.RelO
14:57:22 [http-nio-9080-exec-2] DEBUG o.apache.calcite.plan.RelO
14:57:22 [http-nio-9080-exec-2] DEBUG o.apache.calcite.plan.RelO
14:57:22 [http-nio-9080-exec-2] DEBUG o.apache.calcite.plan.RelO
14:57:22 [http-nio-9080-exec-2] DEBUG o.apache.calcite.plan.RelO
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.t.c.CalciteQuery
FROM
(SELECT `imp_date` AS `sys_imp_date`, 1 AS `pv`
FROM
s2_pv_uv_statis) AS `t3`
WHERE `sys_imp_date` >= '2024-07-08' AND `sys_imp_date` <= '2024
GROUP BY `sys_imp_date`
LIMIT 365]
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.t.DetailQueryOpt
FROM
s2_pv_uv_statis)
SELECT sys_imp_date, SUM(pv) AS pv FROM t_1 WHERE sys_imp_date >
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.t.DetailQueryOpt
FROM
s2_pv_uv_statis)
SELECT sys_imp_date, SUM(pv) AS pv FROM t_1 WHERE sys_imp_date >
14:57:22 [http-nio-9080-exec-2] DEBUG o.s.w.s.m.m.a.RequestRespo
14:57:22 [http-nio-9080-exec-2] DEBUG o.s.w.s.m.m.a.RequestRespo
14:57:22 [http-nio-9080-exec-2] DEBUG o.s.web.servlet.Dispatcher
14:57:22 [http-nio-9080-exec-6] DEBUG o.s.web.servlet.Dispatcher
14:57:22 [http-nio-9080-exec-6] DEBUG o.s.w.s.m.m.a.RequestMappi
14:57:22 [http-nio-9080-exec-6] DEBUG o.s.w.s.m.m.a.RequestRespo
14:57:22 [http-nio-9080-exec-6] DEBUG c.t.s.h.server.aspect.DimV
14:57:22 [http-nio-9080-exec-6] INFO c.t.s.h.s.f.s.i.S2Semantic
14:57:22 [http-nio-9080-exec-6] DEBUG c.t.s.h.s.f.s.i.S2Semantic
14:57:22 [http-nio-9080-exec-6] DEBUG c.t.s.h.c.cache.CaffeineCa
14:57:22 [http-nio-9080-exec-6] INFO c.t.s.h.core.cache.Default
14:57:22 [http-nio-9080-exec-6] DEBUG c.t.s.h.s.utils.QueryReqCo
14:57:22 [http-nio-9080-exec-6] DEBUG c.t.s.h.s.utils.QueryReqCo

```

```

14:57:22 [http-nio-9080-exec-6] DEBUG c.t.s.h.s.utils.QueryReqCo
14:57:22 [http-nio-9080-exec-6] DEBUG c.t.s.h.s.utils.QueryReqCo
14:57:22 [http-nio-9080-exec-6] DEBUG c.t.s.h.s.utils.QueryReqCo
14:57:22 [http-nio-9080-exec-6] DEBUG c.t.s.h.c.u.SysTimeDimensi
14:57:22 [http-nio-9080-exec-6] DEBUG c.t.s.h.c.u.SysTimeDimensi
14:57:22 [http-nio-9080-exec-6] DEBUG c.t.s.h.c.u.SysTimeDimensi
14:57:22 [http-nio-9080-exec-6] DEBUG c.t.s.h.c.u.SysTimeDimensi
14:57:22 [http-nio-9080-exec-6] DEBUG c.t.s.h.c.u.SysTimeDimensi
14:57:22 [http-nio-9080-exec-6] INFO c.t.s.h.core.executor.Jdbc
14:57:22 [http-nio-9080-exec-6] ERROR c.alibaba.druid.filter.Fil
14:57:22 [http-nio-9080-exec-6] WARN c.alibaba.druid.pool.Druid
14:57:22 [http-nio-9080-exec-6] INFO c.alibaba.druid.pool.Druid
14:57:22 [http-nio-9080-exec-6] DEBUG o.s.b.f.xml.XmlBeanDefinit
14:57:22 [http-nio-9080-exec-6] DEBUG o.s.b.f.s.DefaultListableB
14:57:22 [http-nio-9080-exec-6] DEBUG o.s.b.f.s.DefaultListableB
14:57:22 [http-nio-9080-exec-6] DEBUG o.s.b.f.s.DefaultListableB
14:57:22 [http-nio-9080-exec-6] DEBUG o.s.b.f.s.DefaultListableB
14:57:22 [http-nio-9080-exec-6] DEBUG o.s.b.f.s.DefaultListableB
14:57:22 [http-nio-9080-exec-6] DEBUG o.s.b.f.s.DefaultListableB
14:57:22 [http-nio-9080-exec-6] DEBUG o.s.b.f.s.DefaultListableB
14:57:22 [http-nio-9080-exec-6] DEBUG o.s.b.f.s.DefaultListableB
14:57:22 [http-nio-9080-exec-6] DEBUG o.s.b.f.s.DefaultListableB
14:57:22 [http-nio-9080-exec-6] DEBUG o.s.b.f.s.DefaultListableB
14:57:22 [http-nio-9080-exec-6] DEBUG o.s.b.f.s.DefaultListableB
14:57:22 [http-nio-9080-exec-6] DEBUG o.s.b.f.s.DefaultListableB
14:57:22 [http-nio-9080-exec-6] DEBUG o.s.j.support.SQLErrorCode
14:57:22 [http-nio-9080-exec-6] DEBUG o.s.jdbc.core.JdbcTemplate
FROM
s2_pv_uv_statistic
GROUP BY `imp_date`, `imp_date`)
SELECT SUM(pv) FROM t_1 WHERE (sys_imp_date >= '2024-07-31' AND
14:57:22 [http-nio-9080-exec-6] DEBUG o.s.jdbc.datasource.DataSo
14:57:22 [ForkJoinPool.commonPool-worker-20] DEBUG c.t.s.h.c.cac
FROM
s2_pv_uv_statistic
GROUP BY `imp_date`, `imp_date`)
SELECT SUM(pv) FROM t_1 WHERE (sys_imp_date >= '2024-07-31' AND
14:57:22 [http-nio-9080-exec-6] DEBUG c.t.s.h.core.cache.Default
14:57:22 [ForkJoinPool.commonPool-worker-27] DEBUG c.t.s.h.s.p.m
14:57:22 [ForkJoinPool.commonPool-worker-27] DEBUG c.t.s.h.s.p.m
14:57:22 [http-nio-9080-exec-6] DEBUG c.t.s.c.s.p.m.C.insert 135

```

```
14:57:22 [http-nio-9080-exec-6] DEBUG c.t.s.c.s.p.m.C.insert 135
14:57:22 [ForkJoinPool.commonPool-worker-27] DEBUG c.t.s.h.s.p.m
14:57:23 [http-nio-9080-exec-6] DEBUG c.t.s.c.s.p.m.C.updateLast
14:57:23 [http-nio-9080-exec-6] DEBUG c.t.s.c.s.p.m.C.updateLast
14:57:23 [http-nio-9080-exec-6] DEBUG c.t.s.c.s.p.m.C.updateLast
14:57:23 [http-nio-9080-exec-6] DEBUG org.mybatis.spring.SqlSess
14:57:23 [http-nio-9080-exec-6] DEBUG o.s.w.s.m.m.a.RequestRespo
14:57:23 [http-nio-9080-exec-6] DEBUG o.s.w.s.m.m.a.RequestRespo
14:57:23 [http-nio-9080-exec-6] DEBUG o.s.web.servlet.Dispatcher
14:57:23 [http-nio-9080-exec-5] DEBUG o.s.web.servlet.Dispatcher
14:57:23 [http-nio-9080-exec-5] DEBUG o.s.w.s.m.m.a.RequestMappi
14:57:23 [http-nio-9080-exec-5] DEBUG c.t.s.c.s.p.mapper.ChatMap
14:57:23 [http-nio-9080-exec-5] DEBUG c.t.s.c.s.p.mapper.ChatMap
14:57:23 [http-nio-9080-exec-5] DEBUG c.t.s.c.s.p.mapper.ChatMap
14:57:23 [http-nio-9080-exec-5] DEBUG org.mybatis.spring.SqlSess
14:57:23 [http-nio-9080-exec-5] DEBUG o.s.w.s.m.m.a.RequestRespo
14:57:23 [http-nio-9080-exec-5] DEBUG o.s.w.s.m.m.a.RequestRespo
14:57:23 [http-nio-9080-exec-5] DEBUG o.s.web.servlet.Dispatcher
14:58:00 [scheduling-1] DEBUG c.t.s.h.s.task.DictionaryReloadTas
```

## 详细分析：

### 查询语句推荐

```
14:56:57 [http-nio-9080-exec-10] DEBUG o.s.web.servlet.Dispathe
14:56:57 [http-nio-9080-exec-10] DEBUG o.s.w.s.m.m.a.RequestMapp
14:56:57 [http-nio-9080-exec-10] DEBUG o.s.w.s.m.m.a.RequestResp
```

- 接收请求：
  - 时间戳：14:56:57
  - 服务器线程：`http-nio-9080-exec-10`
  - 动作：接收到一个POST请求，路径为 `/api/chat/query/search`，没有附加任何URL参数（`parameters={}`）。
- 映射请求到处理器：

- **动作**：Spring框架的 `RequestMappingHandlerMapping` 组件负责将接收到的请求映射到具体的处理方法。请求被映射到了

`com.tencent.supersonic.chat.server.rest.ChatQueryController` 类的 `search` 方法。

- **读取请求体**：

- **内容类型**：请求体的内容类型是 "application/json;charset=UTF-8"，意味着传送的数据为JSON格式，字符集为UTF-8。
- **请求体内容**：请求体被读取并解析为一个 `ChatParseReq` 对象，这是一个Java类，用于封装请求中的数据。具体数据包括：
  - `queryText`："超音速对比所有人" —— 请求中包含的查询文本。
  - `chatId`：1 —— 聊天会话的标识符。
  - `agentId`：1 —— 代理或用户的标识符。
  - `topN`：10 —— 请求返回的最多结果数量。
  - `user`：null —— 用户信息未提供。
  - `queryFilters`：null —— 查询过滤器未提供。
  - `saveAns`：(truncated) —— 日志被截断，未显示完全，可能包含是否保存答案的信息等。

```
14:56:57 [ForkJoinPool.commonPool-worker-19] DEBUG c.h.h.c.trie.
14:56:57 [ForkJoinPool.commonPool-worker-19] DEBUG c.h.h.c.trie.
14:56:57 [http-nio-9080-exec-10] DEBUG c.t.s.h.s.f.s.i.RetrieveS
14:56:57 [http-nio-9080-exec-10] DEBUG c.t.s.h.s.f.s.i.RetrieveS
14:56:57 [http-nio-9080-exec-10] DEBUG c.t.s.h.s.f.s.i.RetrieveS
14:56:57 [http-nio-9080-exec-10] DEBUG c.t.s.h.s.f.s.i.RetrieveS
14:56:57 [http-nio-9080-exec-10] INFO c.t.s.h.s.f.s.i.RetrieveS
14:56:57 [http-nio-9080-exec-10] DEBUG c.t.s.h.s.f.s.i.RetrieveS
```

## 1. 文本节点遍历：

- **时间戳**：14:56:57
- **线程**：`ForkJoinPool.commonPool-worker-19`
- **操作**：
  - **动作**：在字典树（Trie）中遍历节点。
  - **状态变化**：在执行 `walkNode` 方法前后的节点信息变化。



- 数据：

- `before`：遍历前的标记为 `[_2_4_metric]`。
- `name`：当前处理的词条为“人均访问次数”。
- `after` 和 `natures`：遍历后的标记保持为 `[_2_4_metric]`，表示这是一个度量指标。

## 2. 文本搜索入口记录：

- 线程：`http-nio-9080-exec-10`
- 动作：
  - 解析搜索文本与识别的实体段（segment），并尝试将文本映射到数据库的相关度量和维度。
  - 文本：原始文本是“超音速对比所有”，检测到的关键词是“人”。
  - 解析结果：返回了一个 `HanlpMapResult` 对象，包含词性为 `_1_4_metric`，这表明文本被解析为度量类型的数据，没有具体的偏移和相似性分值。

## 3. 数据集搜索与映射：

- 操作：
  - 尝试从可用的数据集中找到匹配的度量和维度。
  - 确认没有可用的数据集（`possibleDataSets` 是空的），数据集统计信息显示没有匹配的度量或维度数据。
  - 最终解析结果显示，成功匹配到度量类型，对应的类别ID为1。
- 搜索结果：
  - 推荐查询结果为“超音速对比所有人均访问次数”，具体推荐的度量为“人均访问次数”，数据集名称为“超音数数据集”，模型ID为1，元素类型为度量（METRIC），未完全完成匹配（`isComplete=false`）。

```
14:56:57 [http-nio-9080-exec-10] DEBUG o.s.w.s.m.m.a.RequestResp
14:56:57 [http-nio-9080-exec-10] DEBUG o.s.w.s.m.m.a.RequestResp
14:56:59 [http-nio-9080-exec-3] DEBUG o.s.web.servlet.Dispatcher
14:56:59 [http-nio-9080-exec-3] DEBUG o.s.w.s.m.m.a.RequestMappi
14:56:59 [http-nio-9080-exec-3] DEBUG o.s.w.s.m.m.a.RequestRespo
14:56:59 [http-nio-9080-exec-3] DEBUG o.s.web.servlet.Dispatcher
14:56:59 [http-nio-9080-exec-3] DEBUG o.s.w.s.m.m.a.RequestMappi
14:56:59 [http-nio-9080-exec-3] DEBUG o.s.w.s.m.m.a.RequestRespo
```

- 处理JSON响应体：

- 时间戳：14:56:57
- 线程：`http-nio-9080-exec-10`
- 操作：
  - 动作：使用 'application/json' 响应格式，因为请求头中指定了接受 'application/json', 'text/plain', '/'，而服务器支持 'application/json' 和 'application/\*+json'。
  - 数据写入：写入响应体 `ResultData(code=200, msg=success, data=[SearchResult(recommend=超音速对比所有人均访问次数, subRecommend=人均访问次数, m(truncated)...]`，这表明服务器成功处理了请求，返回了状态码200和成功消息，以及具体的搜索结果数据。

- 接收新的POST请求：

- 时间戳：14:56:59
- 线程：`http-nio-9080-exec-3`
- 重复操作：两次记录相同的步骤，处理新的API请求。
  - 动作：
    - 接收一个新的POST请求到 `"/api/chat/query/search"`，没有明确的参数。
    - 映射到 `ChatQueryController#search` 方法进行处理，这是一个用于搜索的控制器方法。
    - 数据读取：读取请求体，内容为 `ChatParseReq(queryText=超音速对比所有人访问, chatId=1, agentId=1, topN=10, user=null, queryFilters=null, saveA(truncated)...]`，显示正在尝试处理一个包含查询文本和其他搜索参数的请求。

```
14:56:59 [ForkJoinPool.commonPool-worker-12] DEBUG c.h.h.c.trie.  
14:56:59 [ForkJoinPool.commonPool-worker-12] DEBUG c.h.h.c.trie.  
14:56:59 [ForkJoinPool.commonPool-worker-12] DEBUG c.h.h.c.trie.  
14:56:59 [ForkJoinPool.commonPool-worker-12] DEBUG c.h.h.c.trie.  
14:56:59 [ForkJoinPool.commonPool-worker-12] DEBUG c.h.h.c.trie.  
14:56:59 [ForkJoinPool.commonPool-worker-12] DEBUG c.h.h.c.trie.  
14:56:59 [ForkJoinPool.commonPool-worker-12] DEBUG c.h.h.c.trie.  
14:56:59 [ForkJoinPool.commonPool-worker-12] DEBUG c.h.h.c.trie.
```

具体的操作是在一个名为 `BaseNode` 的二叉树（bintrie）结构中对各个指标进行搜索和更新。这里是详细的步骤解释：

## 1. 节点遍历操作：

- 时间戳：14:56:59
- 线程：`ForkJoinPool.commonPool-worker-12`
- 详细步骤：
  - 每一步都涉及到遍历节点（walkNode）的动作，显示遍历之前和之后的状态，以及节点的名字和相关性质（natures）。
  - 遍历前的状态和遍历后的状态表明了节点在遍历前后的指标标识符，如 `[_2_3_metric]`。
  - 节点名称：
    - 访问人数：遍历到与访问人数相关的节点。
    - 访问时长：遍历到与访问时长相关的节点。
    - 访问次数：遍历到与访问次数相关的节点。
    - 访问用户数：再次遍历到与访问人数相关的节点，可能表示这是一个循环遍历或者多次检查相同的指标。

## 2. 性质（Natures）：

- 每次遍历都记录了节点的性质，如 `[_2_3_metric]`，这可能代表了节点的某种特定类型或分类标识，用于在数据结构中进行特定的搜索或操作。

```
14:56:59 [http-nio-9080-exec-3] DEBUG c.t.s.h.s.f.s.i.RetrieveSe
14:56:59 [http-nio-9080-exec-3] DEBUG c.t.s.h.s.f.s.i.RetrieveSe
14:56:59 [http-nio-9080-exec-3] DEBUG c.t.s.h.s.f.s.i.RetrieveSe
14:56:59 [http-nio-9080-exec-3] DEBUG c.t.s.h.s.f.s.i.RetrieveSe
14:56:59 [http-nio-9080-exec-3] DEBUG c.t.s.h.s.f.s.i.RetrieveSe
14:56:59 [http-nio-9080-exec-3] DEBUG c.t.s.h.s.f.s.i.RetrieveSe
14:56:59 [http-nio-9080-exec-3] DEBUG c.t.s.h.s.f.s.i.RetrieveSe
14:56:59 [http-nio-9080-exec-3] INFO c.t.s.h.s.f.s.i.RetrieveSe
14:56:59 [http-nio-9080-exec-3] DEBUG c.t.s.h.s.f.s.i.RetrieveSe
```

- 搜索文本解析：
  - 时间戳：14:56:59
  - 线程：`http-nio-9080-exec-3`

- **详细操作：**
  - 解析传入的查询文本“超音速对比所有人”和检测段“访问”。
  - 返回多个HanlpMapResult结果，其中包括与各种度量标准（指标）相关的性质（natures），例如访问次数、访问时长等。
- **度量和维度的搜索：**
  - 检索与“访问”相关的各个度量和维度，并根据这些度量和维度进行搜索。
  - 解析结果包括各种度量的信息，如访问人数、访问时长等，每个都与模型ID和度量类型（METRIC）关联。
  - 结果显示没有可用的数据集（`possibleDataSets:[]`），数据集统计信息显示没有任何度量或维度数据集。
- **推荐结果：**
  - 提供基于查询的推荐结果，例如推荐“超音速对比所有人访问人数”等。
  - 每个推荐都关联到一个模型（如“超音数数据集”）和模型ID。
- **搜索结果信息：**
  - 输出搜索结果的信息，包括各个推荐的详细描述，如访问人数、访问时长、访问次数等。
  - 显示这些指标相关联的模型、数据集和是否完整的状态（`isComplete=false`）。

```
14:56:59 [http-nio-9080-exec-3] DEBUG o.s.w.s.m.m.a.RequestRespo
14:56:59 [http-nio-9080-exec-3] DEBUG o.s.w.s.m.m.a.RequestRespo
14:56:59 [http-nio-9080-exec-3] DEBUG o.s.web.servlet.Dispatcher
14:57:00 [scheduling-1] DEBUG c.t.s.h.s.task.DictionaryReloadTas
14:57:00 [scheduling-1] DEBUG c.t.s.h.s.w.s.impl.DictWordService
14:57:00 [scheduling-1] DEBUG c.t.s.h.s.w.s.impl.DictWordService
14:57:00 [scheduling-1] DEBUG c.t.s.h.s.w.s.impl.DictWordService
14:57:00 [scheduling-1] DEBUG c.t.s.h.s.w.s.impl.DictWordService
14:57:00 [scheduling-1] DEBUG c.t.s.h.s.w.s.impl.DictWordService
14:57:00 [scheduling-1] DEBUG c.t.s.h.s.w.s.impl.DictWordService
14:57:00 [scheduling-1] DEBUG c.t.s.h.s.w.s.impl.DictWordService
14:57:00 [scheduling-1] DEBUG c.t.s.h.s.task.DictionaryReloadTas
```

## 1. 响应数据处理：

- **时间戳：**14:56:59
- **线程：**`http-nio-9080-exec-3`

- 操作内容：

- 使用 `application/json` 作为响应格式，根据客户端接受的格式（`application/json`, `text/plain`, `/`）和服务端支持的格式（`application/json`, `application/*+json`）进行选择。
- 返回成功的响应数据（状态码200，消息为success），其中包含搜索结果（如“超音速对比所有人访问人数”和子推荐“访问人数”）。

## 2. 完成响应：

- 记录HTTP请求处理完成，响应状态为200 OK，表示客户端请求成功被处理并返回了预期的结果。

## 3. 知识库更新任务：

- 时间戳：14:57:00
- 线程：`scheduling-1`
- 详细操作：
  - 启动知识库更新任务（`reloadKnowledge start`），用于更新系统中的词汇和数据定义。
  - 统计不同类别的词性数量，包括维度（DIMENSION，数量为15）、度量（METRIC，数量为18）、实体（ENTITY，数量为4）、值（VALUE，数量为0）、术语（TERM，数量为6）。
  - 检查并确认字典未被重新加载（`Dictionary hasn't been reloaded.`），这可能是因为没有检测到必要的更新或变更。
  - 完成知识库更新任务（`reloadKnowledge end`），记录任务结束。

```
14:57:02 [http-nio-9080-exec-8] DEBUG o.s.web.servlet.Dispatcher
14:57:02 [http-nio-9080-exec-8] DEBUG o.s.w.s.m.m.a.RequestMappi
14:57:02 [http-nio-9080-exec-8] DEBUG o.s.w.s.m.m.a.RequestRespo
14:57:02 [ForkJoinPool.commonPool-worker-5] DEBUG c.h.h.c.trie.b
14:57:02 [ForkJoinPool.commonPool-worker-5] DEBUG c.h.h.c.trie.b
14:57:02 [ForkJoinPool.commonPool-worker-5] DEBUG c.h.h.c.trie.b
14:57:02 [ForkJoinPool.commonPool-worker-5] DEBUG c.h.h.c.trie.b
14:57:02 [ForkJoinPool.commonPool-worker-5] DEBUG c.h.h.c.trie.b
14:57:02 [ForkJoinPool.commonPool-worker-5] DEBUG c.h.h.c.trie.b
14:57:02 [ForkJoinPool.commonPool-worker-5] DEBUG c.h.h.c.trie.b
```

- 接收和映射请求：

- 时间戳：14:57:02
- 线程：`http-nio-9080-exec-8`
- 操作内容：
  - 接收了一个POST请求到 `/api/chat/query/search` 端点，没有特定的查询参数。
  - 将该请求映射到 `com.tencent.supersonic.chat.server.rest.ChatQueryController#search` 方法，这意味着请求被定向到处理搜索的控制器方法。
- 请求体解析：
  - 在处理请求过程中，服务器从请求体中读取了 `application/json;charset=UTF-8` 格式的数据，内容包括查询文本、会话ID、代理ID等信息，关于超音速对比所有人访问次数的查询请求。
- 数据结构操作：
  - 时间戳：14:57:02
  - 线程：`ForkJoinPool.commonPool-worker-5`
  - 详细操作：
    - 执行名为 `walkNode` 的操作，该操作在二叉字典树（trie）中遍历节点。
    - 遍历的每一步中，程序记录节点遍历前后的状态和相关的词性标签，例如：
      - 遍历前状态为 `[_2_1_metric]`，对应的名称为“访问次数”，显示节点之后的状态也是 `[_2_1_metric]`，表明这是一个与度量相关的节点。
      - 接下来，处理“数次问访”，词性变为 `[_2_1_suffix_metric]`，这可能代表对度量单位的后缀处理或是特定的数据处理逻辑。
      - 最后，处理“数次问访均人”，相关词性为 `[_2_4_suffix_metric]`，进一步处理与这些度量相关的后缀。

```

14:57:02 [http-nio-9080-exec-8] DEBUG c.t.s.h.s.f.s.i.RetrieveSe
14:57:02 [http-nio-9080-exec-8] DEBUG c.t.s.h.s.f.s.i.RetrieveSe
14:57:02 [http-nio-9080-exec-8] DEBUG c.t.s.h.s.f.s.i.RetrieveSe
14:57:02 [http-nio-9080-exec-8] DEBUG c.t.s.h.s.f.s.i.RetrieveSe
14:57:02 [http-nio-9080-exec-8] DEBUG c.t.s.h.s.f.s.i.RetrieveSe
14:57:02 [http-nio-9080-exec-8] DEBUG c.t.s.h.s.f.s.i.RetrieveSe
14:57:02 [http-nio-9080-exec-8] INFO c.t.s.h.s.f.s.i.RetrieveSe
14:57:02 [http-nio-9080-exec-8] DEBUG c.t.s.h.s.f.s.i.RetrieveSe
14:57:02 [http-nio-9080-exec-8] DEBUG o.s.w.s.m.m.a.RequestRespo

```



```
14:57:02 [http-nio-9080-exec-8] DEBUG o.s.w.s.m.m.a.RequestRespo
14:57:02 [http-nio-9080-exec-8] DEBUG o.s.web.servlet.Dispatcher
14:57:04 [http-nio-9080-exec-2] DEBUG o.s.web.servlet.Dispatcher
```

- **搜索处理：**

- **时间戳：**14:57:02
- **线程：**`http-nio-9080-exec-8`
- **操作内容：**
  - `RetrieveServiceImpl` 服务接收到一个搜索请求，内容包括请求中的注册文本和检测到的段落。`HanlpMapResult`提供了词性（metrics，即度量）和匹配度（similarity）。这些结果标识了请求中包含的度量类型，如访问次数和人均访问次数。
  - 分析搜索结果，根据检测到的度量和维度，关联到特定的数据集和模型。例如，访问次数和人均访问次数被关联到模型1，这是一个度量型元素。

- **搜索结果构建：**

- 搜索和维度处理服务将搜索结果整合并准备用于响应客户端请求。这些结果被封装在 `ResultData` 对象中，并设置为状态码200，表示请求成功处理。

- **响应客户端：**

- 使用 `application/json` 格式响应客户端，给出了支持的内容类型，并返回搜索结果，如超音速对比所有人访问次数和人均访问次数等推荐结果。

- **完成响应：**

- 请求成功完成，并返回HTTP状态码200，标示请求已正确处理并响应。

- **接收新的请求：**

- 在 `http-nio-9080-exec-2` 线程上，服务器接收到一个新的POST请求到 `/api/chat/query/parse` 端点，这表明服务器正在处理新的解析请求。

## 关键词识别映射

```
14:57:04 [http-nio-9080-exec-2] DEBUG o.s.w.s.m.m.a.RequestMappi
14:57:04 [http-nio-9080-exec-2] DEBUG o.s.w.s.m.m.a.RequestRespo
14:57:04 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.mapper.BaseMatch
14:57:04 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.mapper.BaseMatch
14:57:06 [http-nio-9080-exec-2] DEBUG c.t.s.h.chat.mapper.BaseMa
14:57:06 [http-nio-9080-exec-2] DEBUG c.t.s.h.chat.mapper.BaseMa
14:57:06 [http-nio-9080-exec-2] DEBUG c.t.s.h.chat.mapper.BaseMa
```

```
14:57:06 [http-nio-9080-exec-2] DEBUG c.t.s.h.chat.mapper.BaseMa
14:57:06 [http-nio-9080-exec-2] DEBUG c.t.s.h.chat.mapper.BaseMa
```

- 请求接收与处理：

- 时间戳：14:57:04
- 线程：`http-nio-9080-exec-2`
- 操作内容：
  - 请求被映射到 `ChatQueryController#parse` 方法，处理一个含查询文本的请求，目标是解析查询文本"超音速对比所有人访问次数"。
  - 解析请求体内容为JSON格式，包括查询文本、会话ID、用户信息等。

## 日志分析

### 1. 线程名称

- `[http-nio-9080-exec-2]`：这是日志的线程信息，表明该日志来自一个处理 HTTP 请求的线程。`nio` 表示使用非阻塞 I/O 模式，`9080` 是监听的端口号，`exec-2` 是线程池中编号为 2 的线程。

### 2. 日志的各个部分

- `DEBUG o.s.w.s.m.a.RequestMappingHandlerMapping 522 - Mapped to com.tencent.supersonic.chat.server.rest.ChatQueryController#parse`
  - `RequestMappingHandlerMapping`：Spring MVC 中的 `RequestMappingHandlerMapping` 类负责将 HTTP 请求映射到控制器的方法。日志表明，接收到的 HTTP 请求被映射到 `ChatQueryController` 的 `parse` 方法。
  - `ChatQueryController#parse(ChatParseReq, HttpServletRequest, HttpServletResponse)`：请求被映射到 `ChatQueryController` 的 `parse` 方法，该方法接收三个参数：`ChatParseReq`（表示传入的请求数据），`HttpServletRequest` 和 `HttpServletResponse`（分别表示 HTTP 请求和响应）。
- `DEBUG o.s.w.s.m.a.ResponseBodyMethodProcessor 91 - Read "application/json;charset=UTF-8" to [ChatParseReq(queryText=超音速对比所有人访问次数, chatId=1, agentId=1, topN=10, user=null, queryFilters=null, save(truncated)...]`
  - `ResponseBodyMethodProcessor`：这个类负责处理请求体和响应体的序列化和反序列化。日志显示，系统从请求中读取了一个 JSON 数据，并将其转换为 `ChatParseReq` 对象。

- `ChatParseReq(queryText=超音速对比所有人访问次数, chatId=1, agentId=1, topN=10, user=null, queryFilters=null)`：这显示了传入的请求内容。请求体包含查询文本（queryText），以及其他一些字段。这里的 `user` 是 `null`，表明用户信息还没有设置，通常在接下来的步骤中会由 `UserHolder.findUser` 方法进行设置。
- `DEBUG c.t.s.h.c.mapper.BaseMatchStrategy 150 - word:访问次数,nature:_1_1_metric,frequency:100000`
  - `BaseMatchStrategy`：这个类的日志表明，系统正在解析查询文本。日志中 `word:访问次数` 说明解析到的词是“访问次数”，这个词的性质被标识为 `_1_1_metric`，其出现频率为 `100000`。
- `DEBUG c.t.s.h.c.mapper.BaseMatchStrategy 43 - terms:[S2Term(word=访问次数,nature=_1_1_metric, offset=8, frequency=100000)],,detectDataSetIds:[]`
  - `terms`：这是系统在查询解析过程中识别出的术语列表。这里识别到一个 `S2Term`，表示该术语是“访问次数”，性质为 `_1_1_metric`，偏移量为 `8`，频率为 `100000`。
  - `detectDataSetIds`：这里显示的检测到的数据集ID列表为空，表示当前解析过程中还没有匹配到特定的数据集。

## 与代码的对应关系

- 控制器 `ChatQueryController` 的 `parse` 方法

这个方法会接收一个 `ChatParseReq` 对象，然后通过 `UserHolder.findUser(request, response)` 获取当前用户并设置到请求中。随后，它调用 `chatService.performParsing(chatParseReq)` 来执行解析逻辑。

- `BaseMatchStrategy`

在解析的过程中，`BaseMatchStrategy` 类似处理如何匹配查询文本中的词汇和系统中的指标（metrics）。日志显示的 `word:访问次数` 就是这个过程的一部分。

- 请求与响应

日志显示了一个典型的请求处理流程：从接收 HTTP 请求，映射到控制器方法，读取并解析请求体，到最后的解析逻辑。这是一个标准的基于 Spring MVC 的请求处理流程。

## 总结

展示了一个HTTP请求从接收到解析的整个过程。首先，请求被映射到 `ChatQueryController` 的 `parse` 方法；然后，系统读取并转换请求体为 `ChatParseReq` 对象，随后 `BaseMatchStrategy` 开始对请求中的查询文本进行解析，识别出查询中的指标“访问次数”。

- **查询关键词映射：**
  - 关键词"访问次数"被识别并与数据集相关联，无额外的数据集ID检测到。
  - 经过关键词映射处理，为各个关键词如"访问次数"、"人均访问次数"和"访问用户数"，识别相关的数据集信息和度量。
- **结果构建与响应准备：**
  - 处理关键词后，生成相关的数据集和模型信息，如度量的名称、业务名称、使用计数、类型、描述等。
  - 对查询中的词汇进行过滤映射，修饰和优化搜索结果，以更精确地匹配用户的查询意图。
  - 继续进行实体映射，这涉及进一步细化和确保查询结果的准确性。
- **日志记录和结束：**
  - 对完成的映射和处理过程进行详细的日志记录，包括处理前后的映射信息和成本分析。
  - 完成处理，服务器准备好将结果以JSON格式响应给客户端。

## 生成初步S2SQL并修正（开始大模型生成过程）

```
14:57:06 [http-nio-9080-exec-2] DEBUG c.t.s.h.s.w.s.i.WorkflowSe
14:57:07 [http-nio-9080-exec-2] INFO c.t.s.h.chat.parser.llm.LL
14:57:19 [http-nio-9080-exec-2] DEBUG c.t.s.h.s.w.s.i.WorkflowSe
14:57:19 [http-nio-9080-exec-2] DEBUG c.t.s.h.s.w.s.i.WorkflowSe
14:57:19 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.c.BaseSemanticCo
14:57:19 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.c.BaseSemanticCo
14:57:19 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.c.BaseSemanticCo
14:57:19 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.c.BaseSemanticCo
14:57:19 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.c.BaseSemanticCo
14:57:19 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.c.BaseSemanticCo
14:57:19 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.c.BaseSemanticCo
14:57:19 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.c.BaseSemanticCo
```

## JSON展开解释：

### 结构化JSON

### WorkflowServiceImpl.java

- 接收和处理查询：
  - 时间戳：14:57:06
  - 线程：`http-nio-9080-exec-2`
  - 操作内容：
    - 利用 `RuleSqlParser` 生成基于规则的SQL解析结果，根据用户请求构建查询策略，这包括识别查询涉及的数据集、模型ID和用户信息等。
- 深度语义解析：
  - 利用 `LLMSqlParser` 进行深度学习模型的解析，尝试将自然语言转换成SQL语句，使用OpenAI的GPT-3.5-turbo模型，目的是提高查询的准确性和相关性。
  - 这包括对查询进行更精细的理解，例如识别度量、维度、和需要聚合的数据字段。
- 查询类型解析：
  - `QueryTypeParser` 分析查询的类型，如分组查询（GROUP BY）或度量模型（METRIC MODEL），以决定如何最有效地处理和响应查询。

## `QueryContext` 类解析

`QueryContext` 类主要负责保存与查询相关的上下文信息，它包含了一些关键字段，如 `queryText`（用户输入的查询文本）、`dataSetIds`（数据集的 ID 集合）、`candidateQueries`（候选查询列表）、`semanticSchema`（语义模式）等。它的主要作用是查询解析和执行提供必要的上下文信息。

关键字段说明：

- **queryText**：用户输入的自然语言查询文本。
- **dataSetIds**：与查询相关的数据集的 ID。
- **modelIdToDataSetIds**：模型 ID 与数据集 ID 的映射关系。
- **user**：当前执行查询的用户信息。
- **text2SQLType**：表示将文本转化为 SQL 的方式（如通过规则或 LLM）。
- **candidateQueries**：候选的语义查询列表，通过解析后生成的可能查询。
- **semanticSchema**：语义模式，用于查询解析时的语义理解。
- **workflowState**：当前查询的工作流状态。
- **llmConfig**：与大语言模型（LLM）相关的配置。
- **exemplars**：示例 SQL，用于支持或参考生成 SQL 的过程。

`getCandidateQueries` 方法根据某种排序标准（通常是解析评分）对候选查询进行排序，并限制返回的数量。这说明在 `QueryTypeParser` 执行前，`QueryContext` 已经包含多个经过不同程度解析的候选查询。

## ChatContext 类解析

`ChatContext` 类则用于保存聊天上下文信息，通常是与用户交互相关的内容。它包括查询的 ID、查询文本以及 `SemanticParseInfo`（语义解析信息）。

关键字段说明：

- **chatId**：聊天的 ID，用于唯一标识一个会话。
- **queryText**：用户的查询文本。
- **parseInfo**：包含了语义解析的详细信息（如生成的 SQL、解析评分等）。
- **user**：执行查询的用户。

这个类的主要目的是跟踪与特定聊天会话相关的查询解析状态。

## QueryTypeParser 类

`QueryTypeParser` 类的主要功能是通过进一步解析和分类，确定每个查询的类型（如 METRIC、TAG、ID 或 DETAIL）。

在上下文中：

- **初始化 S2SQL**：在 `parse` 方法中，首先通过调用 `semanticQuery.initS2Sql()` 初始化每个候选查询的 SQL。这一步依赖于 `QueryContext` 中的 `semanticSchema` 和 `user` 信息。`initS2Sql` 会使用这些上下文信息生成初步的 SQL 查询。
- **设置查询类型**：通过调用 `getQueryType()` 方法，`QueryTypeParser` 对 SQL 查询进行分类。`getQueryType()` 方法会根据 `SqlInfo` 中的信息来判断查询类型。这个过程中使用了 `SemanticSchema` 来检查 `select` 和 `where` 子句中的字段是否与度量（metric）、实体（entity）、标签（tag）或时间维度（time dimension）相关。

结合上下文，我们可以看到：

- `QueryContext` 提供了丰富的上下文信息，如候选查询、语义模式、用户信息等，供 `QueryTypeParser` 在解析过程中使用。
- `ChatContext` 提供了与当前查询相关的会话信息，并记录了初步解析的结果，便于 `QueryTypeParser` 进一步分类。

## QueryTypeParser 类中的 parse 方法



```

@Override
public void parse(QueryContext queryContext, ChatContext chatContext) {
    List<SemanticQuery> candidateQueries = queryContext.getCandidateQueries();
    User user = queryContext.getUser();

    for (SemanticQuery semanticQuery : candidateQueries) {
        // 1. init S2SQL
        semanticQuery.initS2Sql(queryContext.getSemanticSchema(), user);
        // 2. set queryType
        QueryType queryType = getQueryType(queryContext, semanticQuery);
        semanticQuery.getParseInfo().setQueryType(queryType);
    }
}

```

- **数据流**： `QueryContext` 中的 `candidateQueries` 被逐一处理，每个查询都被初始化 S2SQL，随后通过 `getQueryType()` 进行类型分类。
- **候选查询的处理**：在 `QueryContext` 中，候选查询是通过某种排序机制生成的，`QueryTypeParser` 进一步对这些查询进行分类，确定其类型。
- **SQL校正和优化**：
  - **时间戳**：14:57:19
  - 在接下来的步骤中，通过多个语义校正器（如 `WhereCorrector`，`GroupByCorrector`，`AggCorrector` 等），对初步生成的SQL语句进行校正和优化。
  - `supersonic\headless\chat\src\main\java\com\tencent\supersonic\headless\chat\corrector`
  - 这些校正步骤包括确保SQL语句在语法和逻辑上的正确性，例如添加WHERE子句以确保日期范围正确，调整聚合函数和分组条件等，以确保查询结果的准确性和相关性。
- **最终SQL输出**：
  - 输出修正后的SQL语句，确保它完全符合业务需求和数据查询的正确性。

```
14:57:19 [http-nio-9080-exec-2] DEBUG c.t.s.h.s.w.s.impl.SchemaS
14:57:19 [http-nio-9080-exec-2] DEBUG c.t.s.h.s.w.s.impl.SchemaS
14:57:20 [http-nio-9080-exec-2] DEBUG c.t.s.h.s.utils.QueryReqCo
14:57:20 [http-nio-9080-exec-2] DEBUG c.t.s.h.s.utils.QueryReqCo
14:57:20 [http-nio-9080-exec-2] DEBUG c.t.s.h.s.utils.QueryReqCo
14:57:20 [http-nio-9080-exec-2] DEBUG c.t.s.h.s.utils.QueryReqCo
14:57:20 [http-nio-9080-exec-2] DEBUG c.t.s.h.s.utils.QueryReqCo
```

## SQL转化展开解释：

### 详细SQL转化

- 时间戳：14:57:19
  - 线程：`http-nio-9080-exec-2`
  - 操作：
    - 调用 `SchemaServiceImpl` 但未找到统计信息（`statInfos:[]` 和 `typeIdAndStatPair: {}`），可能表明当前没有相关统计数据或该部分的数据集为空。
- SQL字段和表名转换：
  - 使用 `QueryReqConverter` 转换SQL语句中的字段和表名：
    - 原始SQL语句：`SELECT SUM(访问次数) FROM 超音数数据集 WHERE (数据日期 >= '2024-07-31' AND 数据日期 <= '2024-08-06')`
    - 字段名转换：将字段名"访问次数"转换为业务名"pv"。
    - 表名转换：将表名"超音数数据集"转换为系统内部表名"t\_1"。
    - 最终SQL语句：`SELECT SUM(pv) FROM t_1 WHERE (sys_imp_date >= '2024-07-31' AND sys_imp_date <= '2024-08-06')`
- SQL语句优化：
  - 时间戳：14:57:20
  - 检查并确认SQL语句中的排序和聚合别名没有冲突或重复，保证SQL语句的正确性和执行效率。
- 结构化查询请求构造：
  - 构造结构化的查询请求（`queryStructReq`），其中包含如下详细信息：
    - `supersonic\chat\server\src\main\java\com\tencent\supersonic\chat\server\util`
      - 数据集ID：1（对应内部转换后的表名"t\_1"）

- **聚合信息**：聚合字段为"pv"，未指定聚合函数（显示为UNKNOWN），没有更多的聚合参数或别名。
- **日期信息**：使用BETWEEN模式指定日期范围从"2024-07-31"到"2024-08-06"。
- **限制**：查询结果限制为最多2000条记录。
- **缓存信息**：指示查询结果可以缓存，以提高相同或相似查询的响应速度。

```
14:57:20 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.u.SysTimeDimensi
14:57:20 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.u.SysTimeDimensi
14:57:20 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.u.SysTimeDimensi
14:57:20 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.u.SysTimeDimensi
14:57:20 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.u.SysTimeDimensi
14:57:20 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.t.DefaultSemanti
14:57:20 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.t.DefaultSemanti
14:57:20 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.t.DefaultSemanti
14:57:20 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.t.DefaultSemanti
```

## 日志展开解释：

### 日志展开解释

- **添加系统时间维度 ( `SysTimeDimensionBuilder` ):**
  - 在构建时间维度前后，记录了时间维度和其他维度（如部门和页面）的状态。
  - 初始维度包括部门和页面，后续增加了以时间为基础的维度，如日（`sys_imp_date`）、周（`sys_imp_week`）、月（`sys_imp_month`）等，这些都使用特定的日期格式（`yyyy-MM-dd`）和表达式来处理和表示时间数据。
- **时间维度表达式构建:**
  - 使用不同的时间表达式来构建时间维度，例如将具体的日期转换为对应的周或月的开始日期，这有助于后续的数据聚合和分析。
- **SQL查询转换 ( `QueryReqConverter` ):**
  - 将业务名称转换为系统内部的标识，例如将业务字段"访问次数"转换为内部名称"pv"，并将数据集名从"超音数数据集"转换为内部表名"t\_1"。
  - 对最终的SQL查询进行优化，例如调整日期字段名和日期范围条件，以符合系统的查询需求。

- 语义转换器应用 ( `DefaultSemanticTranslator` ):

### SemanticTranslator

- 使用多个语义转换器来处理查询参数，如 `DefaultDimValueConverter` 和 `SqlVariableParseConverter`，以确保SQL查询在语义上准确无误。
  - 最终形成的查询参数显示了聚合操作（如求和），但没有具体的函数定义，显示为UNKNOWN。
- 生成最终SQL查询:
    - 将查询参数和转换结果合并生成最终的SQL查询，这在日志中表示为一个具体的查询语句，用于从表 `t_1` 中查询指定日期范围内的 `pv` 总和。

```
14:57:20 [http-nio-9080-exec-2] INFO c.t.s.h.c.t.DefaultSemanti
14:57:20 [http-nio-9080-exec-2] INFO c.t.s.h.c.t.DefaultSemanti
14:57:20 [http-nio-9080-exec-2] INFO c.t.s.h.c.t.c.s.node.DataS
14:57:20 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.t.c.s.node.DataS
14:57:20 [http-nio-9080-exec-2] DEBUG org.apache.calcite.sql.par
14:57:22 [http-nio-9080-exec-2] DEBUG org.apache.calcite.sql2rel
LogicalProject(sys_imp_date=[0], pv=[2])
  LogicalAggregate(group=[{0, 1}], s2_pv_uv_statis_pv=[SUM($2)])
    LogicalProject(sys_imp_date=[4], imp_date=[4], s2_pv_uv_st
      LogicalTableScan(table=[[s2_pv_uv_statis]])
```

(前四行见上一块分析内嵌文章内)

- 语义解析 ( `DefaultSemanticTranslator` ):
  - 解析数据集查询参数，形成了一个SQL查询语句，该查询计划从名为 `t_1` 的表中选取特定日期范围内的 `pv` 字段的总和。同时确认表支持 `WITH` 语句和别名使用。
  - 解析指标查询，确认查询涉及的指标为 `pv`，使用的维度为 `sys_imp_date`，并且指出这是一个聚合查询，但没有特定的限制条件或排序。
- 数据源措施评估 ( `DataSourceNode` ):
  - 列出数据源措施，这包括不同统计数据的指标如用户部门 ( `user_department` )、访问次数和用户统计 ( `s2_pv_uv_statis` )、停留时间统计 ( `s2_stay_time_statis` ) 等。这些措施用于评估和选择最适合的数据源和统计方法。
  - 进行基础数据源匹配，以确认所有条件均得到满足。
- SQL解析器和查询计划生成 ( `org.apache.calcite` ):

- 使用Apache Calcite框架，进行SQL解析，这包括处理日期函数 `DAYOFWEEK`，用于计算周的开始日期。
- 生成关系代数表达式 (RelNode)，展示了最终的查询执行计划。这个计划包括从名为 `s2_pv_uv statis` 的表中聚合数据，具体操作为对 `pv` 进行求和，并按 `sys_imp_date` 和 `imp_date` 进行分组，表明这是一个按时间和页面聚合统计访问次数的查询。

## Apache Calcite 原理解析：

### 主要功能：

#### 1. SQL 解析和验证：

- Calcite 可以解析标准 SQL 语法，并将其转换为内部的抽象语法树 (AST) 或者关系表达式 (RelNode)。
- 它还可以验证 SQL 查询的合法性，例如检查语法错误、检查字段或表的存在性。

#### 2. 查询优化：

- Calcite 提供了一套优化规则，可以对查询进行逻辑优化。比如它会将查询中的某些操作重写为更高效的形式，或者消除冗余的计算。
- 它支持自定义的优化规则，允许开发者根据特定需求进行扩展。

#### 3. 查询计划生成：

- Calcite 将优化后的查询表达式转换为物理查询计划，即将查询转换为可在特定数据库引擎或执行环境中运行的形式。

#### 4. 多数据源支持：

- Calcite 支持对多种异构数据源的访问，如关系型数据库、NoSQL 数据库、文件系统等，并可以将它们统一为一个 SQL 接口。

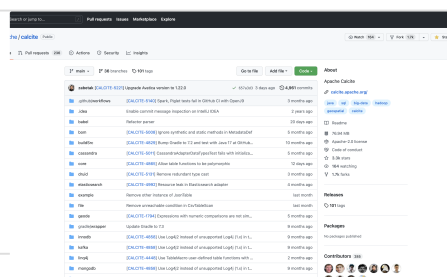
### 其他资料：

#### Apache Calcite 快速入门指南

注意：本文基于 Calcite 1.35.0 版本源码进行学习研究，其他版本可能会存在实现逻辑差异，对源码感兴趣的读者请注意版本选择。

Calcite 简介 Apache Calcite 是一个动态数据管理框架，提供了：


 <https://strongduanmu.com/blog/apache-calcite-quick-start-guide.html>



Apache Calcite教程-SQL解析-Calcite SQL解析\_java使用apcahe calcite 解析sql-CSDN博客

文章浏览阅读1.1w次，点赞9次，收藏24次。Calcite SQL解析代码目录

config.fmpparserImpls.ftl/compoundIdentifier.ftl生成解析器的流程Sql解析使用解析示例代码解析流程常用类  
SpanSqlAbstractParserImplSqlParseExceptionSqlParserSqlParserImplFactorySqlParserPosSqlParserUtilS

 <https://blog.csdn.net/QXC1281/article/details/89481346>

segmentfault.com

<https://segmentfault.com/a/1190000043345921>

<https://segmentfault.com/a/1190000040829100>

## Calcite相关日志解析：

### Calcite相关日志

## 查询优化（Apache Calcite框架）

```
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.t.c.sql.node.Sem
FROM
s2_pv_uv_statist
GROUP BY `imp_date`, `imp_date`
14:57:22 [http-nio-9080-exec-2] DEBUG o.a.c.p.A.rule_execution_s
14:57:22 [http-nio-9080-exec-2] DEBUG o.a.c.p.A.rule_execution_s
Rules
* Total

14:57:22 [http-nio-9080-exec-2] DEBUG o.apache.calcite.plan.RelO
14:57:22 [http-nio-9080-exec-2] DEBUG o.apache.calcite.plan.RelO
14:57:22 [http-nio-9080-exec-2] DEBUG o.apache.calcite.plan.RelO
14:57:22 [http-nio-9080-exec-2] DEBUG o.apache.calcite.plan.RelO
```

这段日志描述了一个SQL查询在Apache Calcite框架中从解析到关系代数优化计划的转换过程，具体内容如下：

- **语义节点优化 ( SemanticNode ):**
  - 执行一个优化的SQL查询，目的是从 `s2_pv_uv_statist` 表中选择 `imp_date` 作为 `sys_imp_date`，并计算总数（这里用 `SUM(1)` 表示，可能是对记录的数量进行统



计)，然后按 `imp_date` 分组。这里的 `GROUP BY` 子句使用相同的字段两次可能是由于日志截断或者某种内部表示。

- **规则执行总结** (`o.a.c.p.A.rule_execution_summary`):

- 提供了HepPlanner（一种优化规划器）执行规则的总结，显示在此过程中没有尝试任何优化规则（`Attempts 0`，`Time 0 us`），表明这个查询没有触发额外的优化规则。

- **最终查询计划生成** (`o.apache.calcite.plan.RelOptPlanner`):

- 显示了多个关系代数表达式（`RelNode`）的使用情况，这些表达式分别表示最终的查询计划中的不同操作：
  - `LogicalProject`：应用投影操作，可能是进行字段选择或转换。
  - `LogicalAggregate`：执行聚合操作，例如这里的 `SUM`。
  - `LogicalProject`：再次执行投影操作，可能用于进一步的数据处理或准备输出格式。
  - `LogicalTableScan`：从 `s2_pv_uv_statis` 表中扫描数据，作为查询的数据源。

```
14:57:22 [http-nio-9080-exec-2] DEBUG org.apache.calcite.sql.par
14:57:22 [http-nio-9080-exec-2] DEBUG org.apache.calcite.sql2rel
LogicalAggregate(group=[{}], EXPR$0=[SUM($0)])
  LogicalProject(pv=[$1])
    LogicalFilter(condition=[AND(>=($0, _UTF-8'2024-07-31'), <=(
      LogicalProject(sys_imp_date=[$0], pv=[$1])
        LogicalAggregate(group=[{0}], pv=[SUM($1)])
          LogicalProject(imp_date=[$4], $f1=[1])
            LogicalTableScan(table=[[s2_pv_uv_statis]])

14:57:22 [http-nio-9080-exec-2] DEBUG org.apache.calcite.sql2rel
LogicalAggregate(group=[{}], EXPR$0=[SUM($0)])
  LogicalProject(pv=[$1])
    LogicalFilter(condition=[AND(>=($0, _UTF-8'2024-07-31'), <=(
      LogicalProject(sys_imp_date=[$0], pv=[$1])
        LogicalAggregate(group=[{0}], pv=[SUM($1)])
          LogicalProject(imp_date=[$4], $f1=[1])
            LogicalTableScan(table=[[s2_pv_uv_statis]])
```

这段日志描述了一个SQL查询在Apache Calcite框架中从解析到关系代数优化计划的转换过程，具体内容如下：

- **SQL解析优化:**

- 针对日期字段 `sys_imp_date` 应用条件约束，确保日期在 `2024-07-31` 至 `2024-08-06` 之间。这是通过 `Reduced` 关键字标记的，表示已经对原始的SQL条件进行了简化或优化处理。

- **SQL到关系代数的转换 ( `org.apache.calcite.sql2rel` ):**

- 这部分日志两次记录了相同的转换过程，显示了将SQL节点 (SqlNode) 转换成关系代数节点 (RelNode) 的详细步骤：
  - **LogicalTableScan**：从 `s2_pv_uv_statis` 表中扫描数据。
  - **LogicalProject**：选择 `imp_date` 作为 `sys_imp_date` 和一个常量1作为 `$f1`，这通常用于计数或标记存在的记录。
  - **LogicalAggregate**：对 `imp_date` 进行分组并计算每个日期的总和 (`SUM($1)`)，其中 `$1` 可能指向常量1，表示计算总记录数。
  - **LogicalProject**：将聚合结果的日期和计算的总和投影为 `sys_imp_date` 和 `pv`。
  - **LogicalFilter**：应用过滤条件，确保 `sys_imp_date` 在指定的日期范围内。
  - **LogicalProject**：再次将数据映射到输出字段，可能用于格式化或准备最终输出。
  - **LogicalAggregate**：最终聚合步骤，计算满足条件的总和。

```
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.t.c.sql.node.Sem
FROM
(SELECT `imp_date` AS `sys_imp_date`, SUM(1) AS `pv`
FROM
s2_pv_uv_statis
GROUP BY `imp_date`) AS `t1`
WHERE `sys_imp_date` >= '2024-07-31' AND `sys_imp_date` <= '2024
14:57:22 [http-nio-9080-exec-2] DEBUG o.apache.calcite.plan.RelO
14:57:22 [http-nio-9080-exec-2] DEBUG o.a.c.p.A.rule_execution_s
14:57:22 [http-nio-9080-exec-2] DEBUG o.a.c.p.A.rule_execution_s
Rules
FilterToGroupScanRule
* Total
```

这段日志记录了SQL查询的优化过程，包括优化规则的应用和执行情况：

- **SQL查询优化:**

- SQL查询是从一个子查询中选择 `imp_date` 作为 `sys_imp_date` 并计算每个日期的总和（`SUM(1)` 表示对每个日期的记录计数），再将结果别名为 `t1`。之后，在外层查询中对 `t1` 结果进行筛选，仅选择 `sys_imp_date` 在 `2024-07-31` 至 `2024-08-06` 之间的数据，并计算这些日期的 `pv` 总和。

- **优化规则的应用 ( `RelOptPlanner` ):**

- **Apply rule [FilterToGroupScanRule]:** 应用了一个名为 `FilterToGroupScanRule` 的优化规则，这个规则尝试将过滤操作（WHERE子句）下推到更早的扫描或聚合操作中，以减少处理数据的总量和提高查询效率。
- **call#0:** 这表示优化过程中的第一次调用或尝试，涉及到的相关关系代数节点（`RelNode`）包括 `LogicalFilter` , `LogicalProject` , `LogicalAggregate` , 和 `LogicalProject` 。

- **优化规则执行统计:**

- 规则 `FilterToGroupScanRule` 被尝试了1次，花费了968微秒。这部分提供了对规则执行效率的评估，帮助理解优化过程的性能。

```
14:57:22 [http-nio-9080-exec-2] DEBUG o.apache.calcite.plan.RelO
14:57:22 [http-nio-9080-exec-2] DEBUG o.apache.calcite.plan.RelO
14:57:22 [http-nio-9080-exec-2] DEBUG o.apache.calcite.plan.RelO
14:57:22 [http-nio-9080-exec-2] DEBUG o.apache.calcite.plan.RelO
14:57:22 [http-nio-9080-exec-2] DEBUG o.apache.calcite.plan.RelO
14:57:22 [http-nio-9080-exec-2] DEBUG o.apache.calcite.plan.RelO
14:57:22 [http-nio-9080-exec-2] DEBUG o.apache.calcite.plan.RelO
```

这段日志记录了Apache Calcite查询优化器（`RelOptPlanner`）在构建最终查询执行计划的过程。每一步都涉及到了不同的逻辑操作节点（`RelNode`），每个节点代表了执行SQL查询的一个步骤：

1. `LogicalTableScan` : 扫描数据表 `s2_pv_uv_stat`，这是查询的起点，从这个表中获取原始数据。
2. `LogicalProject` : 应用一个投影操作，选择表中的特定列（这里选择 `imp_date` 列和常数 `1`），为后续的聚合操作准备数据。
3. `LogicalAggregate` : 执行聚合操作，对投影出的数据按 `imp_date` 分组并计算每组的总和。

4. **LogicalProject**: 再次应用投影，可能是对聚合结果的进一步处理或转换。
5. **LogicalFilter**: 应用过滤条件，只选择日期在 2024-07-31 至 2024-08-06 之间的数据。
6. **LogicalProject**: 根据过滤条件后的结果进行最后的投影操作，可能是对数据格式或数据类型的调整。
7. **LogicalAggregate**: 在过滤和投影处理过的数据上执行最终的聚合操作，计算最终的统计结果（如总和）。

```
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.t.c.CalciteQuery
FROM
(SELECT `imp_date` AS `sys_imp_date`, SUM(1) AS `pv`
FROM
s2_pv_uv_statis
GROUP BY `imp_date`) AS `t7`
WHERE `sys_imp_date` >= '2024-07-31' AND `sys_imp_date` <= '2024
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.t.DetailQueryOpt
FROM
s2_pv_uv_statis
GROUP BY `imp_date`, `imp_date`)
SELECT SUM(pv) FROM t_1 WHERE (sys_imp_date >= '2024-07-31' AND
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.t.DetailQueryOpt
FROM
s2_pv_uv_statis
GROUP BY `imp_date`, `imp_date`)
SELECT SUM(pv) FROM t_1 WHERE (sys_imp_date >= '2024-07-31' AND
```

这些日志描述了查询优化器在处理和优化SQL查询的过程：

1. **简化 SQL 查询 (simplifySql)**: 这一步展示了查询的简化版本，其中从 s2\_pv\_uv\_statis 数据表中选择 imp\_date 列，重命名为 sys\_imp\_date，并计算每个 imp\_date 对应的总计数（这里通过 SUM(1) 实现，意味着统计每个日期的记录数），然后将这些数据作为临时表 t7，以便在WHERE子句中过滤日期在 2024-07-31 至 2024-08-06 之间的记录。
2. **优化器处理前的 SQL (before handleNoMetric)**: 显示了在进一步优化之前的SQL查询。这里使用了SQL的WITH语句（也称为公用表表达式），定义了名为 t\_1 的临时表，临时表 t\_1 通过对 s2\_pv\_uv\_statis 表进行分组并计算每个日期的 pv 总和。
3. **优化器处理后的 SQL (after handleNoMetric)**: 显示了处理之后的SQL查询。在这个阶段，查询优化器应用了某些优化规则，但在这个例子中，处理前后的SQL查询没有变

化。最终的查询是从临时表 `t_1` 中选择日期在指定范围内的 `pv` 总和，并限制了结果的最大行数为1000。

## 第二轮查询优化

```
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.s.utils.QueryReqCo
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.s.utils.QueryReqCo
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.s.utils.QueryReqCo
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.s.utils.QueryReqCo
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.s.utils.QueryReqCo
```

这段日志描述了一系列的 SQL 查询转换和优化过程，主要涉及将查询中的字段名称从业务名称转换为在数据库中使用的实际名称，并应用正确的表名和限制条件。具体步骤和目的如下：

1. **转换字段名称** ( `convert name to bizName before/after` ): 这一步展示了在执行查询前后字段名称的转换过程。起始的 SQL 查询使用了原始字段名称如 `department` 和 `pv`。转换过程中，这些名称保持不变，表明它们已经是业务名称或无需转换。
2. **纠正表名** ( `correctTableName after` ): 日志中显示了将原始表名 `超音数数据集` 替换为系统中使用的表名 `t_1`。这是因为在数据库查询时，可能使用逻辑或临时表名来参考物理表，或者为了简化和标准化查询过程。
3. **替换排序和聚合别名** ( `replaceOrderAggSameAlias` ): 这个步骤确认了查询中使用的排序和聚合操作是否正确应用了别名。在这个例子中，没有实际的替换发生，表明原始查询已经正确地设置了别名和聚合函数。
4. **构建查询请求结构** ( `QueryReqConverter queryStructReq` ): 这一步骤将查询请求转换为内部结构化格式，这样的格式包括数据集ID、模型ID、分组、聚合器、排序、维度和度量过滤器、参数以及日期信息等，使得系统可以更有效地解析和执行SQL查询。此结构也包含缓存信息，指示查询结果可能被缓存以优化性能。

```
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.u.SysTimeDimensi
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.u.SysTimeDimensi
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.u.SysTimeDimensi
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.u.SysTimeDimensi
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.u.SysTimeDimensi
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.t.DefaultSemanti
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.t.DefaultSemanti
```

```
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.t.DefaultSemanti
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.t.DefaultSemanti
```

这段日志记录了一系列系统时间维度的添加和转换过程，以及查询参数的构造和优化过程，具体内容和目的如下：

1. **添加系统时间维度** ( `addSysTimeDimension` ): 日志显示在使用 `SysTimeDimensionBuilder` 类前后的维度信息变化。在这个过程中，系统根据需求动态添加了多种时间维度（如按天、按周、按月），以支持不同粒度的时间查询。例如，增加了 `sys_imp_week` 和 `sys_imp_month` 等时间维度，这些维度对应不同的时间格式化表达式，以便在查询中使用。
2. **维度转换** ( `addSysTimeDimension before/after` ): 这显示了在处理过程中维度对象如何被更新或修改，以适应数据库查询的需求。例如，从 `imp_date` 字段派生出不同的时间维度表达式，并将它们作为新的维度添加到系统中。
3. **查询参数构造** ( `SemanticConverter before/after` ): 这部分展示了查询参数从初始状态到接受各种转换器处理后的最终状态。转换器如 `DefaultDimValueConverter` 和 `SqlVariableParseConverter` 负责处理维度值和解析SQL变量，确保查询参数正确无误。
4. **生成最终查询** ( `DataSetQueryParam` 和 `MetricQueryParam` ): 结合上述转换后的参数，构造最终的数据库查询语句。这包括聚合查询（如对 `pv` 字段的求和）和对结果集的限制条件（如时间范围和记录限制），同时还涉及将逻辑表名转换为数据库中的实际表名（如将 `超音数数据集` 映射到 `t_1`）。

整体来看，这段日志详细记录了从时间维度的处理到查询构造的整个过程，目的是为了确保持生成的SQL查询语句既符合业务需求又优化了数据库的执行效率。

```
14:57:22 [http-nio-9080-exec-2] INFO c.t.s.h.c.t.DefaultSemanti
14:57:22 [http-nio-9080-exec-2] INFO c.t.s.h.c.t.DefaultSemanti
14:57:22 [http-nio-9080-exec-2] INFO c.t.s.h.c.t.c.s.node.DataS
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.t.c.s.node.DataS
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.t.c.s.node.DataS
14:57:22 [http-nio-9080-exec-2] INFO c.t.s.h.c.t.c.sql.render.J
14:57:22 [http-nio-9080-exec-2] DEBUG org.apache.calcite.sql.par
14:57:22 [http-nio-9080-exec-2] DEBUG org.apache.calcite.sql.par
```

这段日志描述了数据查询、数据源配置和查询优化的过程，包括查询解析和数据源之间的匹配与整合，具体内容如下：

1. **解析数据集查询** ( `parse dataSetQuery` ): 这部分日志信息记录了解析数据集查询的参数和配置，其中包括指定的SQL查询语句，如 `SELECT sys_imp_date, department, SUM(pv) AS pv FROM t_1 WHERE sys_imp_date >= '2024-07-08' AND sys_imp_date <= '2024-08-06' GROUP BY`

`sys_imp_date, department LIMIT 365`。这表明该查询旨在从名为 `t_1` 的表中聚合数据，按日期和部门分组，并限制输出行数为365。

2. **数据源度量与维度匹配** (`dataSourceMeasures` 和 `linkDataSources`): 这部分显示了不同数据源的度量和维度信息。例如，`user_department` 和 `s2_pv_uv_statis` 两个数据源包含各自的度量和维度，系统在尝试匹配和整合这些数据源以支持更复杂的查询需求。
3. **数据源不匹配维度** (`baseDataSource not match all dimension`): 这表明在尝试将基本数据源与查询需求中的所有维度匹配时，存在不匹配情况。系统需要进一步操作以确保数据源能满足查询的所有维度需求。
4. **数据源链接** (`linkDataSources`): 这部分说明了系统如何链接不同的数据源以构建更全面的数据视图，例如链接 `user_department` 和 `s2_pv_uv_statis` 数据源，以便可以在一个查询中同时访问用户部门和页面访问统计信息。
5. **查询表视图渲染** (`tableView`): 这一步展示了如何通过渲染SQL视图来链接数据源，例如生成一个视图 `src00_user_department_ccf7`，将用户和部门数据结合起来，以便进行联合查询。
6. **日期函数转换** (`Reduced DAYOFWEEK(imp_date) - 2`): 日志中显示了对SQL函数 `DAYOFWEEK` 的调用，这通常用于计算星期几，并对结果进行调整，可能是为了适配特定的业务逻辑或数据处理需求。

总体来说，这段日志详细记录了从数据源配置、查询优化到视图渲染的复杂过程，目的是为了构建一个能满足具体业务需求的数据查询系统。这包括数据聚合、维度匹配和视图生成，以支持高效的数据分析和报告。

```
14:57:22 [http-nio-9080-exec-2] INFO c.t.s.h.c.t.c.sql.render.J
14:57:22 [http-nio-9080-exec-2] DEBUG org.apache.calcite.sql.par
14:57:22 [http-nio-9080-exec-2] DEBUG org.apache.calcite.sql2rel
LogicalProject(sys_imp_date=[3], department=[1], pv=[2])
  LogicalJoin(condition=[=(0, 4)], joinType=[left])
    LogicalProject(user_name=[0], department=[1])
      LogicalTableScan(table=[[s2_user_department]])
    LogicalProject(s2_pv_uv_statis_pv=[1], sys_imp_date=[4], us
      LogicalTableScan(table=[[s2_pv_uv_statis]])

14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.t.c.sql.node.Sem
FROM
(SELECT `user_name`, `department`
FROM
s2_user_department) AS `t`
```



```
LEFT JOIN (SELECT 1 AS `s2_pv_uv_statistic_pv`, `imp_date` AS `sys_
FROM
s2_pv_uv_statistic) AS `t0` ON `t`.`user_name` = `t0`.`user_name`
```

这段日志描述了如何通过SQL查询来链接和渲染两个数据表，具体过程如下：

1. **视图渲染 ( tableView )**: 这一步创建了一个名为 `src00_s2_pv_uv_statistic_d230` 的视图，该视图基于 `s2_pv_uv_statistic` 表，选择了 `imp_date` 作为 `sys_imp_date`，将 `pv` 的值设定为常量 1，并选择了 `user_name` 和 `imp_date` 字段。这个视图的目的是为了后续的连接操作提供一个结构化的数据源。
2. **字段关系转换 ( Reduced )**: 此操作表示日志记录了在视图或查询中进行字段比较的优化过程，这里是将 `src1_user_department` 表的 `user_name` 字段与 `src1_s2_pv_uv_statistic` 表的 `user_name` 字段进行等值连接条件的简化表示。
3. **SQL到关系模型的转换 ( Plan after converting SqlNode to RelNode )**: 这一步是SQL查询的逻辑计划转换，展示了在逻辑层面如何通过连接操作 ( `LogicalJoin` ) 合并 `user_department` 和 `s2_pv_uv_statistic` 两个数据表。连接类型为左连接 ( `left join` )，条件是两表的 `user_name` 字段相等。这表明系统在尝试将用户部门信息与用户活动数据相结合。
4. **优化关系节点 ( RelNode optimize )**: 这部分描述了对关系模型节点 ( `RelNode` ) 进行的优化，以生成最终的SQL查询。优化后的查询从两个子查询中获取数据：一个从 `s2_user_department` 表中选择 `user_name` 和 `department`，另一个从 `s2_pv_uv_statistic` 表中选择页面访问次数 ( `pv` )、用户名称和日期。然后通过左连接这两个子查询，基于用户名称匹配，来获取最终结果，包括系统日期、部门和页面访问次数。

总结来说，这些日志详细记录了如何从两个不同的数据源中通过SQL查询和连接操作抽取和整合数据，以便进行更深入的数据分析和报告。这显示了复杂查询在数据库中的处理过程，包括视图创建、字段比较、逻辑计划构建和查询优化等步骤。

```
14:57:22 [http-nio-9080-exec-2] DEBUG o.a.c.p.A.rule_execution_s
14:57:22 [http-nio-9080-exec-2] DEBUG o.a.c.p.A.rule_execution_s
Rules
* Total

14:57:22 [http-nio-9080-exec-2] DEBUG o.apache.calcite.plan.RelO
14:57:22 [http-nio-9080-exec-2] DEBUG o.apache.calcite.plan.RelO
14:57:22 [http-nio-9080-exec-2] DEBUG o.apache.calcite.plan.RelO
14:57:22 [http-nio-9080-exec-2] DEBUG o.apache.calcite.plan.RelO
14:57:22 [http-nio-9080-exec-2] DEBUG o.apache.calcite.plan.RelO
14:57:22 [http-nio-9080-exec-2] DEBUG o.apache.calcite.plan.RelO
```

```

14:57:22 [http-nio-9080-exec-2] DEBUG org.apache.calcite.sql.par
14:57:22 [http-nio-9080-exec-2] DEBUG org.apache.calcite.sql.par
14:57:22 [http-nio-9080-exec-2] DEBUG org.apache.calcite.sql2rel
LogicalSort(fetch=[365])
  LogicalAggregate(group=[{0, 1}], pv=[SUM($2)])
    LogicalFilter(condition=[AND(>=($0, _UTF-8'2024-07-08'), <=(
      LogicalProject(sys_imp_date=[$3], department=[$1], pv=[$2])
        LogicalJoin(condition=[=($0, $4)], joinType=[left])
          LogicalProject(user_name=[$0], department=[$1])
            LogicalTableScan(table=[[s2_user_department]])
          LogicalProject(s2_pv_uv_statis_pv=[1], sys_imp_date=[$
            LogicalTableScan(table=[[s2_pv_uv_statis]])

14:57:22 [http-nio-9080-exec-2] DEBUG org.apache.calcite.sql2rel
LogicalSort(fetch=[365])
  LogicalAggregate(group=[{0, 1}], pv=[SUM($2)])
    LogicalFilter(condition=[AND(>=($0, _UTF-8'2024-07-08'), <=(
      LogicalProject(sys_imp_date=[$3], department=[$1], pv=[$2])
        LogicalJoin(condition=[=($0, $4)], joinType=[left])
          LogicalProject(user_name=[$0], department=[$1])
            LogicalTableScan(table=[[s2_user_department]])
          LogicalProject(s2_pv_uv_statis_pv=[1], sys_imp_date=[$
            LogicalTableScan(table=[[s2_pv_uv_statis]])

```

这段日志详细描述了一个查询优化过程，其中使用了Calcite框架来执行和优化SQL查询。下面是详细步骤的解释：

1. **规则尝试信息 (Rule Attempts Info for HepPlanner)**: 日志记录了优化计划中使用的各种规则的尝试次数和执行时间。在此例中，没有规则被尝试。
2. **最终计划的构建 (For final plan)**: 日志记录了构建最终查询计划的各个步骤，每个步骤描述了使用的关系代数表达式 (**RelNode**)。这包括表扫描 (**LogicalTableScan**)、项目 (**LogicalProject**)、连接 (**LogicalJoin**)、以及聚合 (**LogicalAggregate**) 操作。
3. **字段比较简化 (Reduced)**: 这一步表示对字段间比较的简化，例如将 `t2.user_name = t3.user_name` 进行优化。
4. **关系模型转换 (Plan after converting SqlNode to RelNode)**: 描述了将SQL节点转换为关系模型节点的过程。这是优化查询的一部分，确保查询在执行时能高效地处理数据。
5. **关系代数计划的细节**:

- **聚合 (LogicalAggregate)**: 根据 `sys_imp_date` 和 `department` 字段进行分组, 计算每组的 `pv` 总和。
- **筛选 (LogicalFilter)**: 应用时间范围过滤, 只选择 `sys_imp_date` 在指定日期范围内的记录。
- **连接 (LogicalJoin)**: 使用左连接将 `s2_user_department` 表和 `s2_pv_uv_statistic` 表连接起来, 连接条件是 `user_name` 字段相等。
- **排序 (LogicalSort)**: 限制输出结果数量为365条记录, 这可能是为了限制返回的数据量。

整体来看, 这些日志详细记录了从数据库表中获取和处理数据的过程, 涉及连接、筛选、聚合和限制结果数量, 以生成特定的业务报告或数据分析结果。这显示了在处理大规模数据查询时数据库查询优化器的作用和效率。

```
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.t.c.sql.node.Sem
FROM
(SELECT `t0`.`sys_imp_date`, `t`.`department`, `t0`.`s2_pv_uv_st
FROM
(SELECT `user_name`, `department`
FROM
s2_user_department) AS `t`
LEFT JOIN (SELECT 1 AS `s2_pv_uv_statistic_pv`, `imp_date` AS `sys_
FROM
s2_pv_uv_statistic) AS `t0` ON `t`.`user_name` = `t0`.`user_name`)
WHERE `t1`.`sys_imp_date` >= '2024-07-08' AND `t1`.`sys_imp_date
GROUP BY `sys_imp_date`, `department`
LIMIT 365
14:57:22 [http-nio-9080-exec-2] DEBUG o.a.c.p.A.rule_execution_s
14:57:22 [http-nio-9080-exec-2] DEBUG o.a.c.p.A.rule_execution_s
Rules
* Total
```

这段日志描述了一条SQL查询的优化过程, 具体步骤如下:

1. **RelNode优化 (RelNode optimize)**: 这条记录显示了一个优化的SQL查询。查询包括一个嵌套的SELECT语句, 其中:
  - **内部SELECT语句**: 连接了 `s2_user_department` 和 `s2_pv_uv_statistic` 两个表。它将用户部门表中的 `user_name` 和 `department` 字段与页面访问统计表进行左连接, 连接条件

是两个表中的 `user_name` 字段相匹配。内部查询还涉及将访问统计表中的 `imp_date` 字段重命名为 `sys_imp_date`，并将统计的访问量记为 `pv`。

- **外部SELECT语句**：从内部查询生成的结果中选择 `sys_imp_date`（系统时间）、`department`（部门）和 `pv`（访问量）的总和，以及应用了日期范围过滤，只选择日期在 `2024-07-08` 至 `2024-08-06` 之间的记录。最后，按 `sys_imp_date` 和 `department` 分组，并限制结果数量为365条。

2. **规则执行摘要 (Rule Attempts Info for HepPlanner)**: 记录了优化计划 (HepPlanner) 中规则尝试的总结，包括尝试的规则数量和执行时间。在这个案例中，没有任何规则被尝试，说明可能没有适用的优化规则或已经达到最优状态。

这段日志体现了在执行复杂查询时数据库查询优化器的角色，用于提高查询效率，确保数据处理在合理的时间内完成，特别是在涉及大数据量和多表连接的情况下。这是数据库性能调优和管理的关键组成部分。

```
14:57:22 [http-nio-9080-exec-2] DEBUG o.apache.calcite.plan.RelOptPlanner
14:57:22 [http-nio-9080-exec-2] DEBUG o.apache.calcite.plan.RelOptPlanner
14:57:22 [http-nio-9080-exec-2] DEBUG o.apache.calcite.plan.RelOptPlanner
14:57:22 [http-nio-9080-exec-2] DEBUG o.apache.calcite.plan.RelOptPlanner
14:57:22 [http-nio-9080-exec-2] DEBUG o.apache.calcite.plan.RelOptPlanner
14:57:22 [http-nio-9080-exec-2] DEBUG o.apache.calcite.plan.RelOptPlanner
14:57:22 [http-nio-9080-exec-2] DEBUG o.apache.calcite.plan.RelOptPlanner
14:57:22 [http-nio-9080-exec-2] DEBUG o.apache.calcite.plan.RelOptPlanner
14:57:22 [http-nio-9080-exec-2] DEBUG o.apache.calcite.plan.RelOptPlanner
```

这段日志记录了一个数据库查询优化过程的详细步骤，使用的是Apache Calcite的规划器 (RelOptPlanner)。每一条DEBUG信息表示了查询计划中的不同阶段和组件。具体步骤包括：

1. **LogicalSort (rel#130)**: 对结果进行排序，并限制结果数量为365。这通常在查询最后执行，确保返回给用户的结果是有序且数量受控的。
2. **LogicalAggregate (rel#128)**: 应用聚合函数，这里使用了 `SUM` 对字段 `pv`（访问量）进行求和，按照 `sys_imp_date` 和 `department` 进行分组。这是计算部门在指定日期范围内的总访问量。
3. **LogicalFilter (rel#126)**: 过滤条件，确保只考虑日期在 `2024-07-08` 至 `2024-08-06` 之间的数据。这帮助减少处理数据的量，提升查询效率。
4. **LogicalProject (rel#124)**: 数据投影，选择需要的字段 `sys_imp_date`（系统时间），`department`（部门）和 `pv`（访问量）。这一步确定了最终输出中包含的列。
5. **LogicalJoin (rel#122)**: 数据连接操作，左连接 `s2_user_department` 和 `s2_pv_uv_statistic` 两个表。连接条件是用户名称匹配，这常用于结合不同数据源中的相关数据。

6. **LogicalTableScan** (rel#88, rel#91): 表扫描操作, 分别对 s2\_user\_department 和 s2\_pv\_uv\_statistic 两个数据表进行扫描。这是查询执行的基础步骤, 涉及从存储系统读取原始数据。

整个过程显示了如何从初步的表扫描开始, 通过各种变换 (过滤、连接、投影、聚合和排序), 最终形成一条优化的SQL执行计划, 用于实际数据库环境中的查询执行。这些操作有助于确保查询尽可能高效, 同时满足业务需求中对数据处理和分析的具体要求。

```
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.t.c.CalciteQuery
FROM
(SELECT `t6`.`sys_imp_date`, `t5`.`department`, `t6`.`s2_pv_uv_s
FROM
(SELECT `user_name`, `department`
FROM
s2_user_department) AS `t5`
LEFT JOIN (SELECT 1 AS `s2_pv_uv_statistic_pv`, `imp_date` AS `sys_
FROM
s2_pv_uv_statistic) AS `t6` ON `t5`.`user_name` = `t6`.`user_name`)
WHERE `t7`.`sys_imp_date` >= '2024-07-08' AND `t7`.`sys_imp_date
GROUP BY `sys_imp_date`, `department`
LIMIT 365]
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.t.DetailQueryOpt
FROM
(SELECT `user_name`, `department`
FROM
s2_user_department) AS `t2`
LEFT JOIN (SELECT 1 AS `s2_pv_uv_statistic_pv`, `imp_date` AS `sys_
FROM
s2_pv_uv_statistic) AS `t3` ON `t2`.`user_name` = `t3`.`user_name`)
SELECT sys_imp_date, department, SUM(pv) AS pv FROM t_1 WHERE sy
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.t.DetailQueryOpt
FROM
(SELECT `user_name`, `department`
FROM
s2_user_department) AS `t2`
LEFT JOIN (SELECT 1 AS `s2_pv_uv_statistic_pv`, `imp_date` AS `sys_
FROM
s2_pv_uv_statistic) AS `t3` ON `t2`.`user_name` = `t3`.`user_name`)
SELECT sys_imp_date, department, SUM(pv) AS pv FROM t_1 WHERE sy
```

这段日志记录显示了一个复杂SQL查询的简化和优化过程，涉及将多个数据表联结、过滤、聚合，以及最终的查询限制。具体步骤如下：

1. **简化SQL查询** ( `CalciteQueryParser` ): 将一个包含多个子查询和联结的复杂SQL查询进行简化。SQL查询从两个数据表 ( `s2_user_department` 和 `s2_pv_uv_statistic` ) 获取数据，其中包含用户部门信息和页面访问统计。
2. **构建临时表** ( `DetailQueryOptimizer` ): 构建一个临时表 `t_1` ，其中包含来自用户部门表和页面访问统计表的联结结果。这一步骤确保了只有匹配的用户数据会被包括在最终结果中。
3. **联结条件**: 使用 `user_name` 字段将 `s2_user_department` 表（部门信息）和 `s2_pv_uv_statistic` 表（访问统计）进行左联结，以确保所有部门数据都被考虑，即使某些部门可能没有对应的访问数据。
4. **聚合和过滤操作**: 在 `sys_imp_date` 字段（日期字段）的范围 `'2024-07-08'` 至 `'2024-08-06'` 内，对每个部门的页面访问进行求和 ( `SUM` ) 聚合操作。
5. **限制结果数量**: 查询结果被限制为最多365条记录，这可能对应于一年中的每一天，如果查询是按天聚合的。
6. **执行查询**: 最终的SQL查询从 `t_1` 临时表中选择日期、部门和聚合的访问次数，满足指定日期范围内的条件，并按部门和日期进行分组。

这一过程展示了在大型或复杂数据库系统中如何进行查询优化，以提高性能并确保查询结果的准确性和相关性。这种类型的优化对于处理大量数据和提供快速响应时间至关重要。

## 第三轮查询优化

```
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.s.utils.QueryReqCo
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.s.utils.QueryReqCo
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.s.utils.QueryReqCo
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.s.utils.QueryReqCo
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.s.utils.QueryReqCo
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.u.SysTimeDimensi
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.u.SysTimeDimensi
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.u.SysTimeDimensi
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.u.SysTimeDimensi
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.u.SysTimeDimensi
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.t.DefaultSemanti
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.t.DefaultSemanti
```



```
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.t.DefaultSemanti
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.t.DefaultSemanti
```

这段日志记录描述了从数据库查询过程的多个步骤，从原始SQL查询的转换、维度的构建到查询参数的最终确定，具体步骤如下：

#### 1. 查询转换 ( `QueryReqConverter` ):

- 将数据集ID为1的查询从原始名称“超音数数据集”转换为业务名称，并替换表名为 `t_1`。
- 此过程中涉及将查询中的字段和条件调整为系统中的业务名称和结构。

#### 2. 维度时间构建 ( `SysTimeDimensionBuilder` ):

- 在时间维度构建器中添加和修改维度设置，包括时间粒度（如日、周、月）和是否为主要维度。这有助于后续的数据聚合和分析。
- 这一步涉及将不同的时间表达式和格式化细节，如日、周、月等，配置成系统可以理解 and 处理的维度。

#### 3. 语义转换器 ( `DefaultSemanticTranslator` ):

- 初始阶段接收查询参数，识别其中包含的聚合器、维度过滤器等。
- 接受特定的转换器如维度值转换器和SQL变量解析转换器，这些转换器负责将查询参数转换为详细的SQL查询。
- 最终形成了一个详细的查询参数结构，这个结构定义了具体的SQL语句、涉及的表、使用的聚合选项等。

#### 4. 查询结构化请求 ( `queryStructReq` ):

- 描述了具体的查询需求，包括数据集ID、模型ID、聚合信息、过滤条件、日期信息等。
- 指定了查询类型为详细查询（DETAIL），并详细指定了日期范围和聚合函数的使用情况。

这些步骤展示了从接收原始查询到生成优化后的SQL执行计划的详细过程，每一步都针对特定的业务和数据结构要求进行了调整和优化，以确保查询的效率和准确性。

```
14:57:22 [http-nio-9080-exec-2] INFO c.t.s.h.c.t.DefaultSemanti
14:57:22 [http-nio-9080-exec-2] INFO c.t.s.h.c.t.DefaultSemanti
14:57:22 [http-nio-9080-exec-2] INFO c.t.s.h.c.t.c.s.node.DataS
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.t.c.s.node.DataS
14:57:22 [http-nio-9080-exec-2] DEBUG org.apache.calcite.sql.par
```



```

14:57:22 [http-nio-9080-exec-2] DEBUG org.apache.calcite.sql.par
14:57:22 [http-nio-9080-exec-2] DEBUG org.apache.calcite.sql.par
14:57:22 [http-nio-9080-exec-2] DEBUG org.apache.calcite.sql2rel
LogicalProject(sys_imp_date=[$4], pv=[1])
  LogicalTableScan(table=[[s2_pv_uv_statis]])

14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.t.c.sql.node.Sem
FROM
s2_pv_uv_statis
14:57:22 [http-nio-9080-exec-2] DEBUG o.a.c.p.A.rule_execution_s
14:57:22 [http-nio-9080-exec-2] DEBUG o.a.c.p.A.rule_execution_s
Rules
* Total

```

这段日志记录描述了SQL查询的解析和优化过程，涉及多个不同的组件和步骤，具体内容和流程如下：

#### 1. 数据集查询解析 ( `DefaultSemanticTranslator` ):

- 将输入的查询参数解析为一个具体的数据集查询请求。查询内容为按日期聚合页面访问量，从数据表 `t_1` 中选择特定日期范围内的数据，应用聚合操作，并限制结果的数量。
- 该查询使用外部聚合选项 ( `OUTER` ) 处理可能的缺失数据，确保包含所有日期范围内的记录。

#### 2. 指标查询解析 ( `DefaultSemanticTranslator` ):

- 解析出指标查询参数，明确了聚合函数是对 `pv` 字段进行求和，并且此查询被标记为本地执行 ( `nativeQuery=true` )，意味着将直接在数据源执行而不进行进一步的转换或优化。

#### 3. 数据源测量 ( `DataSourceNode` ):

- 检查数据源中的测量数据，例如用户部门、页面访问统计和停留时间统计的可用性。此步骤确保所需数据的可用性。

#### 4. SQL解析 ( `org.apache.calcite.sql.parser` ):

- 进行日期字段的函数计算，这里具体为 `DAYOFWEEK(imp_date) - 2`，计算周中日期的偏移。

#### 5. SQL到关系模型转换 ( `org.apache.calcite.sql2rel` ):

- 将SQL语句转换为关系代数表达式。此步骤中，将 `imp_date` 转换为 `sys_imp_date`，并固定 `pv` 值为1，意味着对每个条目计数。

## 6. 关系节点优化 ( `SemanticNode` ):

- 对生成的关系节点进行优化，这通常涉及选择最有效的执行计划或应用变换规则以简化查询。

## 7. 执行计划生成和优化 ( `RelOptPlanner` 和 `rule_execution_summary` ):

- 根据优化器的规则尝试生成最终的执行计划，但此处显示尝试次数为0，表明在此阶段没有应用额外的优化规则。

这些步骤展示了从SQL命令到执行计划的生成过程，涵盖了解析、优化和准备执行的各个方面，确保查询以最优方式执行。

```
14:57:22 [http-nio-9080-exec-2] DEBUG o.apache.calcite.plan.RelO
14:57:22 [http-nio-9080-exec-2] DEBUG o.apache.calcite.plan.RelO
14:57:22 [http-nio-9080-exec-2] DEBUG org.apache.calcite.sql.par
14:57:22 [http-nio-9080-exec-2] DEBUG org.apache.calcite.sql2rel
LogicalSort(fetch=[365])
  LogicalAggregate(group=[{0}], pv=[SUM($1)])
    LogicalFilter(condition=[AND(>=($0, _UTF-8'2024-07-08'), <=
      LogicalProject(sys_imp_date=[$4], pv=[1])
        LogicalTableScan(table=[[s2_pv_uv_statis]])

14:57:22 [http-nio-9080-exec-2] DEBUG org.apache.calcite.sql2rel
LogicalSort(fetch=[365])
  LogicalAggregate(group=[{0}], pv=[SUM($1)])
    LogicalFilter(condition=[AND(>=($0, _UTF-8'2024-07-08'), <=
      LogicalProject(sys_imp_date=[$4], pv=[1])
        LogicalTableScan(table=[[s2_pv_uv_statis]])
```

这段日志记录详细描述了一个SQL查询的转换和优化过程，具体涉及的步骤和操作如下：

### 1. 执行计划选择 ( `RelOptPlanner` ):

- 这部分展示了优化器选择的最终执行计划的不同组成部分。使用了逻辑投影和逻辑表扫描操作符，这表明优化器在构建最终查询执行计划时确定了数据的处理流程。

### 2. 日期过滤简化 ( `org.apache.calcite.sql.parser` ):

- 对SQL中的日期条件进行简化，`sys_imp_date >= '2024-07-08' AND sys_imp_date <= '2024-08-06'`，这是一个典型的日期范围过滤操作，用于限定查询结果只包含特定时间段内的数据。

### 3. SQL节点到关系模型的转换 ( `org.apache.calcite.sql2rel` ):

- 转换过程中, Calcite框架将SQL节点转换为关系代数模型 (RelNode) , 这个过程包括以下几个关键操作:
  - `LogicalSort` : 应用排序操作, 这里的排序限定了最多返回365行数据, 可能用于限制结果集大小。
  - `LogicalAggregate` : 聚合操作, 这里进行的是按 `sys_imp_date` 分组的求和操作, 聚合字段是 `pv` 。
  - `LogicalFilter` : 应用过滤条件, 确保只计算指定日期范围内的数据。
  - `LogicalProject` : 数据投影操作, 选择 `sys_imp_date` 字段并将 `pv` 设置为1, 后者可能是一个统计或者计数操作的标记。
  - `LogicalTableScan` : 表扫描操作, 直接从 `s2_pv_uv_stat` 表中读取数据。

### 4. 重复操作的确认:

- 相同的转换和优化过程被重复记录两次, 这可能是日志系统的冗余记录或确保优化步骤的正确应用。

整体来看, 这些日志记录描绘了一个从SQL解析到执行计划确定的全过程, 体现了在数据库查询优化中涉及的不同技术和方法。这个过程的目标是确保查询能以最有效的方式执行, 同时满足特定的业务需求和性能标准。

```
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.t.c.sql.node.Sem
FROM
(SELECT imp_date AS sys_imp_date, 1 AS pv
FROM
s2_pv_uv_stat) AS t
WHERE sys_imp_date >= '2024-07-08' AND sys_imp_date <= '2024-08-
GROUP BY sys_imp_date
LIMIT 365
14:57:22 [http-nio-9080-exec-2] DEBUG o.a.c.p.A.rule_execution_s
14:57:22 [http-nio-9080-exec-2] DEBUG o.a.c.p.A.rule_execution_s
Rules
* Total
```

这段日志记录描述了一个SQL查询的优化过程, 具体步骤如下:

#### 1. 优化关系模型 (RelNode) :

- 对一个SQL查询进行优化，查询的目的是从 `s2_pv_uv_statis` 表中选取日期和统计值，计算从2024年7月8日到2024年8月6日之间的每一天的 `pv`（页面访问量）总和。
- SQL语句首先将表 `s2_pv_uv_statis` 中的 `imp_date` 重命名为 `sys_imp_date`，并将每一行的 `pv` 值设为1，然后对这些数据按 `sys_imp_date` 进行分组，求每组的 `pv` 总和，限制结果数量为365行。

## 2. HepPlanner的规则执行总结:

- 这一部分日志记录了优化规划器（HepPlanner）在优化过程中的尝试和花费时间。在这个例子中，尽管优化器被激活，但实际上并没有应用任何规则，这从“总尝试次数”为0可以看出。

这段日志提供了关于如何通过SQL优化和执行计划来处理和分析数据的内部视角，显示了数据库管理系统在查询优化阶段的具体操作。这种优化有助于提高查询效率，确保在指定的日期范围内快速准确地汇总数据。

```
14:57:22 [http-nio-9080-exec-2] DEBUG o.apache.calcite.plan.RelO
14:57:22 [http-nio-9080-exec-2] DEBUG o.apache.calcite.plan.RelO
14:57:22 [http-nio-9080-exec-2] DEBUG o.apache.calcite.plan.RelO
14:57:22 [http-nio-9080-exec-2] DEBUG o.apache.calcite.plan.RelO
14:57:22 [http-nio-9080-exec-2] DEBUG o.apache.calcite.plan.RelO
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.t.c.CalciteQuery
FROM
(SELECT `imp_date` AS `sys_imp_date`, 1 AS `pv`
FROM
s2_pv_uv_statis) AS `t3`
WHERE `sys_imp_date` >= '2024-07-08' AND `sys_imp_date` <= '2024
GROUP BY `sys_imp_date`
LIMIT 365]
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.t.DetailQueryOpt
FROM
s2_pv_uv_statis)
SELECT sys_imp_date, SUM(pv) AS pv FROM t_1 WHERE sys_imp_date >
14:57:22 [http-nio-9080-exec-2] DEBUG c.t.s.h.c.t.DetailQueryOpt
FROM
s2_pv_uv_statis)
SELECT sys_imp_date, SUM(pv) AS pv FROM t_1 WHERE sys_imp_date >
```

这段日志详细记录了一个SQL查询优化的过程，目的是从数据库表 `s2_pv_uv_statistic` 中获取数据，并计算2024年7月8日至2024年8月6日每天的访问次数总和（pv）。具体步骤如下：

### 1. 构建基础查询：

- SQL查询首先将 `s2_pv_uv_statistic` 表中的 `imp_date` 列重命名为 `sys_imp_date`，并假设每条记录的 `pv` 值为1。

### 2. 逻辑计划构建：

- **逻辑表扫描**：从 `s2_pv_uv_statistic` 表中获取数据。
- **逻辑投影**：选择 `imp_date` 作为 `sys_imp_date`，并将每条记录的 `pv` 值设置为1。
- **逻辑过滤**：筛选出日期在2024年7月8日至2024年8月6日之间的数据。
- **逻辑聚合**：对 `sys_imp_date` 进行分组，计算每组的 `pv` 总和。
- **逻辑排序**：限制结果为最多365条记录，对应每天一条记录。

### 3. SQL简化和优化：

- 在优化过程中，查询被封装在一个公共表表达式（CTE，即 `WITH` 子句）中，名为 `t_1`，其中执行与上述逻辑相同的步骤。
- 优化器处理查询前后，保持了相同的结构，显示优化过程主要关注在逻辑层面上的数据准备和表达式简化，没有对查询逻辑做出显著变更。

整个过程的目标是有效地从原始数据中抽取、转换并聚合特定日期范围内的数据，以提供快速响应的查询结果。这种优化有助于提升查询性能，特别是在处理大量数据时。

## 开始查询

```
14:57:22 [http-nio-9080-exec-2] DEBUG o.s.w.s.m.m.a.RequestRespo
14:57:22 [http-nio-9080-exec-2] DEBUG o.s.w.s.m.m.a.RequestRespo
14:57:22 [http-nio-9080-exec-2] DEBUG o.s.web.servlet.Dispatcher
14:57:22 [http-nio-9080-exec-6] DEBUG o.s.web.servlet.Dispatcher
14:57:22 [http-nio-9080-exec-6] DEBUG o.s.w.s.m.m.a.RequestMappi
14:57:22 [http-nio-9080-exec-6] DEBUG o.s.w.s.m.m.a.RequestRespo
14:57:22 [http-nio-9080-exec-6] DEBUG c.t.s.h.server.aspect.DimV
14:57:22 [http-nio-9080-exec-6] INFO c.t.s.h.s.f.s.i.S2Semantic
14:57:22 [http-nio-9080-exec-6] DEBUG c.t.s.h.s.f.s.i.S2Semantic
14:57:22 [http-nio-9080-exec-6] DEBUG c.t.s.h.c.cache.CaffeineCa
```

这些日志记录了一个Web应用程序中处理HTTP请求和响应的过程，具体是在使用Spring框架的Web服务器上执行数据库查询和缓存操作。详细步骤如下：

## 1. 内容协商：

- 系统使用 `application/json` 内容类型响应，考虑到请求头提供的类型包括 `application/json` , `text/plain` , `/*` , 系统支持的类型包括 `application/json` , `application/*+json` 。

## 2. 写入响应：

- 系统写入响应体，内容是一个名为 `ResultData` 的对象，包含状态码200（成功），消息为"success"，并包含一个名为 `ParseResp` 的对象，该对象中包括了关于用户聊天查询的一些信息（例如查询ID和文本）。

## 3. 完成请求处理：

- 请求被处理完成，返回状态码200 OK。

## 4. 处理新的POST请求：

- 服务器接收到一个新的POST请求，URL为 `"/api/chat/query/execute"`，没有提供URL参数。

## 5. 映射到控制器：

- 请求被映射到 `ChatQueryController` 类的 `execute` 方法，此方法处理聊天查询执行的请求。

## 6. 读取请求体：

- 从请求中读取 `application/json;charset=UTF-8` 格式的数据，解析成 `ChatExecuteReq` 对象，该对象包含查询执行所需的各种参数，如用户ID、代理ID、查询文本等。

## 7. 执行数据库查询：

- 生成并执行一个SQL查询，目的是从名为 `超音数数据集` 的数据库表中查询2024年7月31日到2024年8月6日之间的访问次数总和。

## 8. 查询请求处理：

- 处理查询请求，将查询请求内容封装为 `QuerySqlReq` 对象，并设置结果的限制为最多1000条记录。

## 9. 缓存操作：

- 生成一个缓存键，用于检查或存储查询结果，以提高处理速度和减少数据库负载。
- 检查缓存中是否存在对应的值，由于缓存值为null，表示缓存中没有现成的数据。

整个流程涉及到接收请求、处理请求、执行数据库查询及缓存处理等，目的是高效地响应用户对聊天数据的查询请求。

```
14:57:22 [http-nio-9080-exec-6] INFO c.t.s.h.core.cache.Default
14:57:22 [http-nio-9080-exec-6] DEBUG c.t.s.h.s.utils.QueryReqCo
14:57:22 [http-nio-9080-exec-6] DEBUG c.t.s.h.s.utils.QueryReqCo
14:57:22 [http-nio-9080-exec-6] DEBUG c.t.s.h.s.utils.QueryReqCo
14:57:22 [http-nio-9080-exec-6] DEBUG c.t.s.h.s.utils.QueryReqCo
14:57:22 [http-nio-9080-exec-6] DEBUG c.t.s.h.s.utils.QueryReqCo
14:57:22 [http-nio-9080-exec-6] DEBUG c.t.s.h.c.u.SysTimeDimensi
14:57:22 [http-nio-9080-exec-6] DEBUG c.t.s.h.c.u.SysTimeDimensi
14:57:22 [http-nio-9080-exec-6] DEBUG c.t.s.h.c.u.SysTimeDimensi
14:57:22 [http-nio-9080-exec-6] DEBUG c.t.s.h.c.u.SysTimeDimensi
14:57:22 [http-nio-9080-exec-6] DEBUG c.t.s.h.c.u.SysTimeDimensi
```

这些日志记录涵盖了一个Web服务中缓存查询、数据集转换和时间维度构建的过程，使用的是Spring框架和一个基于Calcite的SQL优化工具。以下是详细的中文解释：

#### 1. 从缓存查询数据：

- 系统尝试从缓存中查询数据，使用的缓存键为 `supersonic:dev:0:-1:f7b1b5e688abc7568ab130888606780a`。这表明系统在执行查询之前检查缓存，以减少数据库的访问次数并加速响应。

#### 2. 数据集转换：

- 将查询从使用业务名称前的语句转换为使用业务名称后的语句。具体是将 `SELECT SUM(访问次数)` 转换为 `SELECT SUM(pv)`，并将日期字段从 `数据日期` 更改为 `sys_imp_date`。这显示了系统在查询执行前对SQL语句进行了优化和标准化处理。

#### 3. 更正表名：

- 将查询中的表名从业务名称转换为系统内部使用的表名，即从 `超音数数据集` 更改为 `t_1`。

#### 4. 时间维度构建：

- 在执行查询之前，系统动态地添加时间维度到查询中。这包括处理日期字段和其他与时间相关的字段，例如将 `imp_date` 字段转换为不同的时间粒度（如日、周、月），以支持更复杂的时间序列分析。

#### 5. 维度信息调整：

- 对于页面类型和时间类型的字段，如 `imp_date` 和 `page`，系统设置了不创建维度（`isCreateDimension=0`）和创建维度（`isCreateDimension=1`）的标志。这些设置反映了不同字段在数据模型中的使用方式，如是否作为查询的一部分。

整个过程展示了一个高度优化的数据查询和处理环境中，如何利用缓存、动态SQL重写和维度管理来提高数据处理效率和查询性能。这些操作有助于提升系统的响应速度和处理



大规模数据集的能力。

```
14:57:22 [http-nio-9080-exec-6] INFO    c.t.s.h.core.executor.Jdbc
14:57:22 [http-nio-9080-exec-6] ERROR   c.alibaba.druid.filter.Fil
14:57:22 [http-nio-9080-exec-6] WARN    c.alibaba.druid.pool.Druid
14:57:22 [http-nio-9080-exec-6] INFO    c.alibaba.druid.pool.Druid
14:57:22 [http-nio-9080-exec-6] DEBUG   o.s.b.f.xml.XmlBeanDefinit
14:57:22 [http-nio-9080-exec-6] DEBUG   o.s.b.f.s.DefaultListableB
14:57:22 [http-nio-9080-exec-6] DEBUG   o.s.b.f.s.DefaultListableB
14:57:22 [http-nio-9080-exec-6] DEBUG   o.s.b.f.s.DefaultListableB
14:57:22 [http-nio-9080-exec-6] DEBUG   o.s.b.f.s.DefaultListableB
14:57:22 [http-nio-9080-exec-6] DEBUG   o.s.b.f.s.DefaultListableB
14:57:22 [http-nio-9080-exec-6] DEBUG   o.s.b.f.s.DefaultListableB
14:57:22 [http-nio-9080-exec-6] DEBUG   o.s.b.f.s.DefaultListableB
14:57:22 [http-nio-9080-exec-6] DEBUG   o.s.b.f.s.DefaultListableB
14:57:22 [http-nio-9080-exec-6] DEBUG   o.s.b.f.s.DefaultListableB
14:57:22 [http-nio-9080-exec-6] DEBUG   o.s.b.f.s.DefaultListableB
14:57:22 [http-nio-9080-exec-6] DEBUG   o.s.b.f.s.DefaultListableB
14:57:22 [http-nio-9080-exec-6] DEBUG   o.s.b.f.s.DefaultListableB
14:57:22 [http-nio-9080-exec-6] DEBUG   o.s.j.support.SQLErrorCode
```

这段日志记录显示了一个基于Spring框架的Web服务在执行数据库操作时的详细步骤。以下是详细的中文解释：

### 1. 执行SQL查询：

- 系统正在执行一个复杂的SQL查询，其中包含一个临时表 `t_1`。此临时表是根据 `imp_date` 分组，并计算每个日期的页面访问总数（假设为 `1`）。然后从这个临时表中选取特定日期范围（'2024-07-31'至'2024-08-06'）内的数据，计算这段时间内的总访问量，并限制输出结果最多为1000条记录。

### 2. 错误与警告处理：

- `ERROR` 日志指出在加载数据库相关的 `stat` 过滤器时发生错误，提示 "filter not found"，表示系统配置中可能缺少指定的过滤器。
- `WARN` 日志警告称 `removeAbandoned` 属性设置为 `true`，这在生产环境中不推荐使用，因为它可能会影响数据库连接池的稳定性。

### 3. 数据源初始化：

- `DruidDataSource` 表示一个数据库连接池（名为 `数据实例`）已经初始化完毕，准备用于数据库操作。

### 4. Spring Bean的加载和实例化：

- 日志中显示了多个数据库错误代码配置的Bean被加载。这些Bean支持多种数据库（如DB2, Derby, H2, MySQL等），使得Spring框架可以适配这些不同的数据库平台，通过识别不同的错误代码来更好地处理数据库操作中可能出现的异常。

## 5. 数据库错误处理：

- 最后一条日志表示尝试获取数据库 `1@数据实例` 的错误代码时没有找到相应的配置，这可能导致框架无法正确识别和处理该数据库实例中发生的特定错误。

整个过程展示了如何在Java企业应用中配置和使用数据库连接，以及如何通过Spring框架管理数据库错误处理的机制，确保应用可以在遇到数据库错误时做出适当的响应。

```
14:57:22 [http-nio-9080-exec-6] DEBUG o.s.jdbc.core.JdbcTemplate
FROM
s2_pv_uv_statistic
GROUP BY `imp_date`, `imp_date`)
SELECT SUM(pv) FROM t_1 WHERE (sys_imp_date >= '2024-07-31' AND
14:57:22 [http-nio-9080-exec-6] DEBUG o.s.jdbc.datasource.DataSource
14:57:22 [ForkJoinPool.commonPool-worker-20] DEBUG c.t.s.h.c.cache
FROM
s2_pv_uv_statistic
GROUP BY `imp_date`, `imp_date`)
SELECT SUM(pv) FROM t_1 WHERE (sys_imp_date >= '2024-07-31' AND
14:57:22 [http-nio-9080-exec-6] DEBUG c.t.s.h.core.cache.Default
```

这段日志记录反映了一个基于Spring框架的应用在处理数据库查询及缓存操作的具体步骤。具体包括以下几个关键动作：

### 1. 执行SQL查询：

- `JdbcTemplate` 是Spring提供的一个简化数据库交互的工具，这里它正在执行一个SQL查询。查询包含一个临时表 `t_1`，该表通过汇总 `s2_pv_uv_statistic` 表中的 `imp_date` 字段来计算访问次数，再从这个临时表中选取特定日期范围（'2024-07-31'至'2024-08-06'）的数据，计算总访问次数，并限制结果最多返回1000条记录。

### 2. 获取数据库连接：

- `DataSourceUtils` 是Spring用于管理数据库连接的工具，日志中显示正在从数据源获取一个JDBC连接，这是执行任何数据库操作前的必要步骤。

### 3. 缓存操作：

- `CaffeineCacheManager` 是一个高性能的Java缓存库，这里用于将查询结果存入缓存。具体操作是将一个包含SQL查询结果的对象（`SemanticQueryResp`），包括查询产生的列信息和查询语句本身，缓存到键为 `supersonic:dev:0:-1:f7b1b5e688abc7568ab130888606780a` 的缓存项中。
- `DefaultQueryCache` 日志显示了将查询结果放入缓存的操作，这有助于后续相同的查询能够直接从缓存中获取数据，提高系统的响应速度和效率。

这段日志详细记录了从执行数据库查询到将结果缓存的整个过程，显示了数据处理的完整流程和涉及的多个系统组件。

```
14:57:22 [ForkJoinPool.commonPool-worker-27] DEBUG c.t.s.h.s.p.m
14:57:22 [ForkJoinPool.commonPool-worker-27] DEBUG c.t.s.h.s.p.m
14:57:22 [http-nio-9080-exec-6] DEBUG c.t.s.c.s.p.m.C.insert 135
14:57:22 [http-nio-9080-exec-6] DEBUG c.t.s.c.s.p.m.C.insert 135
14:57:22 [ForkJoinPool.commonPool-worker-27] DEBUG c.t.s.h.s.p.m
14:57:23 [http-nio-9080-exec-6] DEBUG c.t.s.c.s.p.m.C.updateLast
14:57:23 [http-nio-9080-exec-6] DEBUG c.t.s.c.s.p.m.C.updateLast
14:57:23 [http-nio-9080-exec-6] DEBUG c.t.s.c.s.p.m.C.updateLast
14:57:23 [http-nio-9080-exec-6] DEBUG org.mybatis.spring.SqlSess
```

这段日志记录详细描述了数据库操作、日志记录、以及数据库交互的多个阶段，涉及到数据插入、查询构建和系统状态更新等多个方面：

### 1. 插入查询统计信息：

- 使用 `createRecord` 方法向 `s2_query_stat_info` 表中插入一条记录，包括跟踪ID、模型ID、数据集ID、用户信息、查询类型等详细参数。这条记录可能用于后续的查询统计和监控。

### 2. 插入聊天信息：

- 使用 `insert` 方法向 `s2_chat_memory` 表中插入一条记录，包括问题、代理ID、数据库架构、执行的SQL语句等信息。此操作可能是记录用户通过聊天界面提交的查询问题，以便进行查询历史记录和审计。

### 3. 更新聊天信息：

- 使用 `updateLastQuestion` 方法更新 `s2_chat` 表，设置 `last_question` 和 `last_time` 字段，这可能是为了记录最后一次聊天的内容和时间，用于用户的聊天历史跟踪。

### 4. 数据库操作的关闭：

- `SqlSessionUtils` 关闭非事务性SqlSession，表示当前的数据库操作已完成并且关闭了数据库连接。

整体上，这些日志显示了在用户通过聊天界面进行交互时后端如何处理和记录查询请求的详细步骤，涵盖了从用户输入处理到数据库操作和状态更新的完整流程。这有助于在需要时追踪问题源头，优化系统性能或进行安全审计。

```
14:57:23 [http-nio-9080-exec-6] DEBUG o.s.w.s.m.m.a.RequestRespo
14:57:23 [http-nio-9080-exec-6] DEBUG o.s.w.s.m.m.a.RequestRespo
14:57:23 [http-nio-9080-exec-6] DEBUG o.s.web.servlet.Dispatcher
14:57:23 [http-nio-9080-exec-5] DEBUG o.s.web.servlet.Dispatcher
14:57:23 [http-nio-9080-exec-5] DEBUG o.s.w.s.m.m.a.RequestMappi
14:57:23 [http-nio-9080-exec-5] DEBUG c.t.s.c.s.p.mapper.ChatMap
14:57:23 [http-nio-9080-exec-5] DEBUG c.t.s.c.s.p.mapper.ChatMap
14:57:23 [http-nio-9080-exec-5] DEBUG c.t.s.c.s.p.mapper.ChatMap
14:57:23 [http-nio-9080-exec-5] DEBUG org.mybatis.spring.SqlSess
14:57:23 [http-nio-9080-exec-5] DEBUG o.s.w.s.m.m.a.RequestRespo
14:57:23 [http-nio-9080-exec-5] DEBUG o.s.w.s.m.m.a.RequestRespo
14:57:23 [http-nio-9080-exec-5] DEBUG o.s.web.servlet.Dispatcher
14:58:00 [scheduling-1] DEBUG c.t.s.h.s.task.DictionaryReloadTas
```

这段日志记录描述了一系列后端服务处理的步骤，涉及响应客户端的HTTP请求，查询数据库，以及服务端返回处理结果的详细信息：

### 1. 处理JSON响应：

- `RequestResponseBodyMethodProcessor` 处理器使用 `application/json` 格式响应客户端的请求，说明客户端接受了多种格式，但是最终选择了JSON格式来传输数据。

### 2. 返回查询结果：

- 服务器成功构建了查询结果并通过HTTP状态码200（成功）返回。具体的返回数据被截断，但提到了关于查询结果的信息，包括查询模式和SQL语句。

### 3. 接收并处理新的GET请求：

- `DispatcherServlet` 接收到一个GET请求，请求路径为 `"/api/chat/manage/getAll?agentId=1"`。这表明请求是为了获取某个代理ID下的所有对话信息。
- `RequestMappingHandlerMapping` 将请求映射到 `ChatController#getAllConversations` 方法上。

### 4. 数据库查询操作：

- `ChatMapper.getAll` 方法准备并执行SQL查询，选出特定代理ID下所有未删除的聊天记录，结果按置顶和最后时间排序。
- 查询参数包括创建者和代理ID，返回了一个结果。

## 5. 关闭数据库会话：

- `SqlSessionUtils` 关闭了非事务性的`SqlSession`，这意味着这个查询不涉及到数据库事务操作。

## 6. 返回聊天记录列表：

- 服务器再次使用 `application/json` 格式响应客户端的GET请求，返回所有相关的聊天记录详情。

## 7. 定时任务启动：

- `DictionaryReloadTask` 的 `reloadKnowledge` 方法被触发，这可能是一个定时任务，用于定期重新加载或更新服务器的知识库或数据字典。

整体来看，这些日志记录了从接收HTTP请求、处理请求、查询数据库到返回数据的完整流程，并且涵盖了对API端点的访问、数据库操作、及其结果的返回，同时还包括系统定期任务的启动。这些信息有助于监控系统的运行状态和调试问题。

## 反复出现的数据库查询操作示例（包含查询指标、聊天记录等）：

```
14:28:27 [http-nio-9080-exec-1] DEBUG org.mybatis.spring.SqlSess
14:28:27 [http-nio-9080-exec-1] DEBUG org.mybatis.spring.SqlSess
14:28:27 [http-nio-9080-exec-1] DEBUG o.s.jdbc.datasource.DataSo
14:28:27 [http-nio-9080-exec-1] DEBUG o.m.s.t.SpringManagedTrans
14:28:27 [http-nio-9080-exec-1] DEBUG c.t.s.c.s.p.m.C.selectList
14:28:27 [http-nio-9080-exec-1] DEBUG c.t.s.c.s.p.m.C.selectList
14:28:27 [http-nio-9080-exec-1] DEBUG c.t.s.c.s.p.m.C.selectList
14:28:27 [http-nio-9080-exec-1] DEBUG c.t.s.c.s.p.m.C.selectList
14:28:27 [http-nio-9080-exec-1] DEBUG c.t.s.c.s.p.m.C.selectList
14:28:27 [http-nio-9080-exec-1] DEBUG c.t.s.c.s.p.m.C.selectList
14:28:27 [http-nio-9080-exec-1] DEBUG org.mybatis.spring.SqlSess
```

这段日志记录展示了一个后端服务在处理HTTP请求过程中与数据库的交互。具体步骤如下：

### 1. 创建新的`SqlSession`：

- `Creating a new SqlSession`：日志显示正在创建一个新的SQL会话（`SqlSession`）。这是与数据库交互的第一步，涉及到建立一个数据库会话环境。

### 2. `SqlSession`注册情况：

- `SqlSession [org.apache.ibatis.session.defaults.DefaultSqlSession@428ba2f6] was not registered for synchronization because synchronization is not active`：这说明创建的SqlSession没有注册同步，因为当前环境没有激活同步操作。这通常意味着该操作不在事务管理之下。

### 3. 获取数据库连接：

- `Fetching JDBC Connection from DataSource`：从数据源获取JDBC连接，这是执行SQL命令前的准备步骤。

### 4. JDBC连接管理：

- `JDBC Connection [com.mysql.cj.jdbc.ConnectionImpl@3404ed53] will not be managed by Spring`：表明获取的JDBC连接不会被Spring框架管理。通常这种情况出现在手动管理数据库连接或在非标准事务环境下操作时。

### 5. 执行SQL查询（计数查询）：

- `Preparing: SELECT count(0) FROM s2_chat_query WHERE (chat_id = ? AND user_name = ?)`：准备执行一个SQL查询，计算符合特定条件（chat\_id和user\_name）的记录数。
- `Parameters: 1(Long), admin(String)`：指明SQL查询中使用的参数，这里使用了聊天ID为1和用户名为"admin"。
- `Total: 1`：查询结果显示，有1条记录满足条件。

### 6. 执行SQL查询（选择列表）：

- `Preparing: SELECT question_id, agent_id, create_time, user_name, query_state, chat_id, score, feedback, query_text, query_result, similar_queries, parse_time_cost FROM s2_chat_query WHERE (chat_id = ? AND user_name = ?) ORDER BY question_id DESC LIMIT ?`：准备执行一个更复杂的SQL查询，选择多个字段，条件同上，并按question\_id降序排列，限制返回结果数为3。
- `Parameters: 1(Long), admin(String), 3(Integer)`：SQL查询使用的参数，包括聊天ID、用户名以及返回的最大记录数。
- `Total: 3`：查询结果显示，有3条记录满足条件。

### 7. 关闭SqlSession：

- `Closing non transactional SqlSession [org.apache.ibatis.session.defaults.DefaultSqlSession@428ba2f6]`：日志记录了非事务性SqlSession的关闭操作，表明此次数据库会话结束。

这些日志条目详细记录了从创建数据库会话到查询执行再到会话关闭的全过程，提供了数据库操作的详细视图，有助于理解后端服务如何处理特定的数据请求。