`ParseInfoProcessor` +Sql `InfoProcessor`

`ParseInfoProcessor`

`ParseInfoProcessor` 类是一个用于处理查询结果并提取结构化信息的处理器类。它实现了 `ResultProcessor` 接口，并在查询执行后更新查询的解析信息 (`SemanticParseInfo`)。

主要功能

1. **处理解析响应** (`process` 方法):
   - 该方法从 `QueryContext` 获取候选查询 (`candidateQueries`)，然后对每个候选查询的 `SemanticParseInfo` 调用 `updateParseInfo` 方法进行解析信息的更新。
2. **更新解析信息** (`updateParseInfo` 方法):
   - 此方法是核心功能，用于解析 SQL 语句并更新 `SemanticParseInfo` 对象。它解析 SQL 中的字段表达式（主要是过滤器和分组条件），然后提取时间信息、维度过滤器、度量和维度信息等。
   - 主要包括以下步骤：
     1. **检查 SQL 是否已经被修正** (`correctS2SQL`):
        - 如果 `correctS2SQL` 与 `S2SQL` 不同，才会进行更新操作。
     2. **提取过滤器表达式**:
        - 从 `correctS2SQL` 中提取出所有的过滤器表达式，并用于后续的时间信息和维度过滤器的提取。
     3. **设置时间信息** (`DateConf`):
        - 从过滤器表达式中提取出时间相关的信息，更新到 `parseInfo` 的 `dateInfo` 字段。
     4. **设置维度过滤器** (`QueryFilter`):
        - 根据提取出的过滤器表达式，匹配相应的 `SchemaElement`（维度），并生成 `QueryFilter`，添加到 `parseInfo` 的 `dimensionFilters` 中。
     5. **获取 SQL 中的所有字段**:

- 提取 `correctS2SQL` 中涉及的所有字段，区分度量和维度，并更新 `parseInfo` 的 `metrics` 和 `dimensions` 字段。

主要辅助方法

1. `getElements`：
   - 该方法用于根据字段名从维度或度量集合中提取匹配的 `SchemaElement`。
2. `getFieldsExceptDate`：
   - 过滤掉时间维度字段，返回其他字段列表。
3. `getDimensionFilter`：
   - 根据字段名和过滤器表达式，生成维度过滤器列表。
4. `getDateInfo`：
   - 从过滤器表达式中提取出时间信息（如日期范围）。
5. `getNameToElement`：
   - 生成一个映射表，将字段名映射到对应的 `SchemaElement`，支持别名。

类的关联关系

1. **与 `SchemaService` 的关联**:
   - `ParseInfoProcessor` 使用 `SchemaService` 来获取语义架构 (`SemanticSchema`)，以便在 SQL 解析时匹配和识别维度与度量。
2. **与 `SemanticParseInfo` 的关联**:
   - 该类的主要工作就是更新 `SemanticParseInfo` 对象，这是语义解析过程中存储结构化信息的核心数据结构。
3. **与 `SqlSelectHelper` 的关联**:
   - 通过 `SqlSelectHelper` 提取 SQL 中的字段信息、过滤器和分组条件，辅助解析 SQL。

总结

`ParseInfoProcessor` 是语义查询系统中的一个关键组件，它负责在查询解析之后对 SQL 进行深入分析，提取并更新结构化的语义信息。这些信息包括时间范围、维度过滤器、度量与维度等，都是进一步处理和展示查询结果的基础。通过与 `SchemaService` 和 `SqlSelectHelper` 的协作，`ParseInfoProcessor` 能够准确地解析和识别 SQL 语句中的关键元素。

`SqlInfoProcessor`

`SqlInfoProcessor` 类是一个用于处理 SQL 相关信息的处理器。它的主要职责是在查询解析的结果中添加 S2SQL（Semantic to SQL）信息，便于技术用户验证生成的 SQL 查询。

主要功能

1. **处理解析响应** (`process` 方法):
   - 该方法从 `QueryContext` 中获取候选查询 (`semanticQueries`)，然后对每个查询的 `SemanticParseInfo` 调用 `addSqlInfo` 方法，以填充 SQL 信息。
2. **添加 SQL 信息** (`addSqlInfo` 方法):
   - 此方法是核心功能，用于生成 SQL 查询并将其添加到 `SemanticParseInfo` 中。
   - 主要步骤包括：
     1. **创建查询对象**: 通过 `QueryManager` 根据 `QueryMode` 创建对应的 `SemanticQuery` 对象。
     2. **构建查询请求**: 调用 `SemanticQuery` 的 `buildSemanticQueryReq` 方法，构建 `SemanticQueryReq` 对象。
     3. **调用解释服务**: 使用 `SemanticLayerService` 的 `explain` 方法生成 SQL 查询，并将结果存储在 `ExplainResp` 对象中。
     4. **记录日志**: 如果查询是通过 `LLMSqlQuery` 生成的，会将结果记录到 `keyPipelineLog` 日志中。
     5. **更新 `SqlInfo` 对象**: 将生成的 SQL 查询 (`querySql`) 和数据源 ID (`sourceId`) 更新到 `SqlInfo` 对象中。

主要方法解析

1. **`process`**:
   - `process` 方法是该类的入口，用于处理整个查询流程。它调用 `addSqlInfo` 为每个候选查询添加 SQL 信息。
2. **`addSqlInfo(QueryContext queryContext, List<SemanticParseInfo> semanticParseInfos)`**:
   - 该方法遍历 `SemanticParseInfo` 列表，并为每个解析信息调用 `addSqlInfo(QueryContext queryContext, SemanticParseInfo parseInfo)` 方法。

- 其主要作用是处理多个候选查询的情况。

3. `addSqlInfo(QueryContext queryContext, SemanticParseInfo parseInfo)`：
   - 这是最核心的方法，用于实际生成并添加 SQL 信息。具体步骤包括：
     - 通过 `QueryManager.createQuery` 方法创建 `SemanticQuery` 对象。
     - 调用生成 SQL 的服务 (`SemanticLayerService.explain`) 并获取 SQL 语句。
     - 将生成的 SQL 信息更新到 `SemanticParseInfo` 的 `SqlInfo` 中。

类的关联关系

1. **与 `QueryContext` 和 `ChatContext` 的关联**:
   - `SqlInfoProcessor` 通过 `QueryContext` 获取候选查询，并使用 `ChatContext` 来跟踪查询的上下文信息。
2. **与 `SemanticLayerService` 的关联**:
   - `SemanticLayerService` 是关键服务，用于生成 SQL 语句。`SqlInfoProcessor` 调用其 `explain` 方法来获取最终的 SQL 查询。
3. **与 `LLMSqlQuery` 的关联**:
   - 如果候选查询是通过 `LLMSqlQuery` 生成的，`SqlInfoProcessor` 会额外记录日志，输出 SQL 信息的细节。
4. **与 `QueryManager` 的关联**:
   - `QueryManager` 用于根据查询模式创建适当的 `SemanticQuery` 对象，这是生成 SQL 查询的关键步骤。

总结

`SqlInfoProcessor` 在语义解析工作流中扮演着将高级查询转换为 SQL 的角色。它确保生成的 SQL 查询能够准确地反映用户的意图，并且为技术用户提供了验证 SQL 的途径。