SqlVariableParseConverter

它的主要功能是根据给定的查询语句（`QueryStatement`）中的参数，解析 SQL 语句中的变量，并将解析后的 SQL 语句设置到相应的数据源中。下面是对代码的详细分析：

类的基本结构

1. `@Slf4j`:
   - 这是一个 Lombok 注解，用于自动生成日志记录器对象 `log`，可以通过 `log.info()`，`log.error()` 等方法进行日志记录。
2. `@Component("SqlVariableParseConverter")`:
   - 这是一个 Spring 注解，用于将这个类注册为 Spring 的组件，并指定组件的名称为 `"SqlVariableParseConverter"`。这样可以通过这个名称在 Spring 容器中引用这个组件。
3. **实现 `QueryConverter` 接口**:
   - 这个类实现了 `QueryConverter` 接口，因此必须提供 `accept` 和 `convert` 两个方法的实现。

`accept` 方法

```
@Override
public boolean accept(QueryStatement queryStatement) {
    if (Objects.isNull(queryStatement.getQueryParam())) {
        return false;
    }
    return true;
}
```

- **作用**:
  - 这个方法用于判断当前转换器是否应该处理传入的 `queryStatement`。如果 `queryStatement` 中的 `QueryParam` 为空，则返回 `false`，表示不处理该查询语句。否则，返回 `true`，表示该转换器可以处理该查询语句。
- **逻辑**:
  - `queryStatement.getQueryParam()` 用于获取查询参数，如果查询参数为空，说明没有要处理的变量或参数，因此不需要进行转换。

`convert` 方法

```
@Override
public void convert(QueryStatement queryStatement) {
    SemanticSchemaResp semanticSchemaResp = queryStatement.getSemanticSc
hemaResp();
    List<ModelResp> modelResps = semanticSchemaResp.getModelResps();
    if (CollectionUtils.isEmpty(modelResps)) {
        return;
```

```
    }
    for (ModelResp modelResp : modelResps) {
        if (ModelDefineType.SQL_QUERY.getName()
                .equalsIgnoreCase(modelResp.getModelDetail().getQueryTyp
e())) {
            String sqlParsed = SqlVariableParseUtils.parse(
                    modelResp.getModelDetail().getSqlQuery(),
                    modelResp.getModelDetail().getSqlVariables(),
                    queryStatement.getQueryParam().getParams()
            );
            DataSource dataSource = queryStatement.getSemanticModel()
                    .getDatasourceMap().get(modelResp.getBizName());
            dataSource.setSqlQuery(sqlParsed);
        }
    }
}
```

- **作用**:
  - 这个方法负责将查询语句中的 SQL 变量解析出来，并替换为实际的值，最后将解析后的 SQL 语句更新到数据源中。
- **具体逻辑**:
  1. **获取语义模式响应**:
     - `SemanticSchemaResp semanticSchemaResp = queryStatement.getSemanticSchemaResp();`
     - 从 `queryStatement` 中获取语义模式响应 (`SemanticSchemaResp`)，它包含了关于语义模式的所有信息。
  2. **获取模型响应列表**:
     - `List<ModelResp> modelResps = semanticSchemaResp.getModelResps();`
     - 获取语义模式响应中的模型响应 (`ModelResp`) 列表。每个 `ModelResp` 表示一个业务模型的详细信息。
  3. **检查模型响应列表是否为空**:
     - `if (CollectionUtils.isEmpty(modelResps)) { return; }`
     - 如果模型响应列表为空，则直接返回，不做进一步处理。
  4. **遍历模型响应**:
     - `for (ModelResp modelResp : modelResps) { ... }`
     - 遍历每个模型响应，检查它们是否定义了 SQL 查询。
  5. **检查模型是否定义了 SQL 查询**:
     - `if (ModelDefineType.SQL_QUERY.getName().equalsIgnoreCase(modelResp.getModelDetail().getQueryType())) { ... }`

- 检查模型的查询类型是否是 `SQL_QUERY` 类型。`ModelDefineType.SQL_QUERY` 是一个枚举值，表示模型是以 SQL 查询形式定义的。如果是，则进入下一个步骤。

6. **解析 SQL 变量**:
   - `String sqlParsed = SqlVariableParseUtils.parse(...);`
   - 使用 `SqlVariableParseUtils` 工具类的方法 `parse` 来解析模型中定义的 SQL 查询。这个方法会根据模型中的 SQL 模板 (`SqlQuery`) 和 SQL 变量 (`SqlVariables`)，结合查询语句中的参数 (`queryStatement.getQueryParam().getParams()`)，生成最终的 SQL 查询。
7. **更新数据源的 SQL 查询**:
   - `DataSource dataSource = queryStatement.getSemanticModel().getDatasourceMap().get(modelResp.getBizName());`
   - `dataSource.setSqlQuery(sqlParsed);`
   - 获取对应模型的 `DataSource` 数据源对象，并将解析后的 SQL 查询设置到这个数据源中。`BizName` 用于查找与该模型关联的数据源。

总结

- **整体流程**:
  1. 检查查询语句是否包含参数。
  2. 获取与查询语句关联的所有业务模型。
  3. 对于每个定义了 SQL 查询的模型，解析其中的 SQL 变量。
  4. 将解析后的 SQL 查询更新到相应的数据源对象中。
- **用途**:
  - 该转换器在查询过程中负责处理 SQL 变量的解析工作，这对于支持动态查询和用户自定义查询非常重要。通过这种方式，系统可以根据用户输入或上下文动态生成 SQL 查询，从而满足各种复杂的查询需求。