DefaultDimValueConverter

负责将查询语句中的维度默认值添加到 SQL 语句的 `WHERE` 子句中。以下是代码的详细分析：

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("DefaultDimValueConverter")`:
   - 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 `"DefaultDimValueConverter"`。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) {
    return !Objects.isNull(queryStatement.getDataSetQueryParam
())
            && !StringUtils.isBlank(queryStatement.getDataSetQue
ryParam().getSql());
}
```

- **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 `DataSetQueryParam` 为空，或者 `SQL` 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.getDataSetQueryParam()` 用于获取数据集查询参数，如果查询参数为空，说明没有要处理的数据集查询参数。
  - `queryStatement.getDataSetQueryParam().getSql()` 获取 SQL 查询字符串，如果为空，说明没有可处理的 SQL。

`convert` 方法

```
@Override
public void convert(QueryStatement queryStatement) {
    List<Dimension> dimensions = queryStatement.getSemanticModel
().getDimensions().stream()
            .filter(dimension -> !CollectionUtils.isEmpty(dimens
ion.getDefaultValues()))
            .collect(Collectors.toList());
    if (CollectionUtils.isEmpty(dimensions)) {
        return;
    }
    String sql = queryStatement.getDataSetQueryParam().getSql();
    List<String> whereFields = SqlSelectHelper.getWhereFields(sq
l)
            .stream().filter(field -> !TimeDimensionEnum.contain
sTimeDimension(field))
            .collect(Collectors.toList());
    if (!CollectionUtils.isEmpty(whereFields)) {
        return;
    }
    MetricTable metricTable = queryStatement.getDataSetQueryPara
m()
            .getTables().stream().findFirst().orElse(null);
    List<Expression> expressions = Lists.newArrayList();
    for (Dimension dimension : dimensions) {
        ExpressionList expressionList = new ExpressionList();
        List<Expression> exprs = new ArrayList<>();
        dimension.getDefaultValues().forEach(value -> exprs.add
(new StringValue(value)));
        expressionList.setExpressions(exprs);
        InExpression inExpression = new InExpression();
        inExpression.setLeftExpression(new Column(dimension.getB
izName()));
        inExpression.setRightExpression(expressionList);
        expressions.add(inExpression);
        if (metricTable != null) {
            metricTable.getDimensions().add(dimension.getBizName
());
        }
    }
    sql = SqlAddHelper.addWhere(sql, expressions);
```

```
        queryStatement.getDataSetQueryParam().setSql(sql);
    }
```

- **作用**:
  - 这个方法负责将维度的默认值添加到 SQL 查询的 `WHERE` 子句中。如果查询语句中的维度具有默认值，并且这些维度没有被 `WHERE` 子句覆盖，则这些默认值将被添加到 SQL 中。
- **具体逻辑**:
  1. **获取带有默认值的维度列表**:
     - `List<Dimension> dimensions = queryStatement.getSemanticModel().getDimensions().stream().filter(dimension -> !CollectionUtils.isEmpty(dimension.getDefaultValues())).collect(Collectors.toList());`
     - 从查询语句的语义模型中获取所有维度，并过滤出那些具有默认值的维度。
  2. **检查维度列表是否为空**:
     - `if (CollectionUtils.isEmpty(dimensions)) { return; }`
     - 如果没有任何维度具有默认值，直接返回，不做进一步处理。
  3. **获取 SQL 查询字符串**:
     - `String sql = queryStatement.getDataSetQueryParam().getSql();`
     - 获取当前的 SQL 查询字符串。
  4. **获取并过滤 `WHERE` 子句中的字段**:
     - `List<String> whereFields = SqlSelectHelper.getWhereFields(sql).stream().filter(field -> !TimeDimensionEnum.containsTimeDimension(field)).collect(Collectors.toList());`
     - 从 SQL 查询中提取 `WHERE` 子句中的字段，并过滤掉时间维度的字段。这里使用了 `TimeDimensionEnum.containsTimeDimension(field)` 来检查字段是否是时间维度。
  5. **检查 `WHERE` 子句中的字段是否为空**:
     - `if (!CollectionUtils.isEmpty(whereFields)) { return; }`
     - 如果 `WHERE` 子句中已经包含了需要的字段，直接返回，不做进一步处理。
  6. **获取 `MetricTable` 对象**:
     - `MetricTable metricTable = queryStatement.getDataSetQueryParam().getTables().stream().findFirst().orElse(null);`
     - 从查询语句中获取 `MetricTable` 对象，这个对象包含了查询中涉及的度量和维度。
  7. **创建并添加 `InExpression` 表达式**:

- 遍历所有带有默认值的维度，为每个维度创建一个 `InExpression` 表达式，这个表达式用于将默认值作为过滤条件添加到 SQL 查询的 `WHERE` 子句中。
- `InExpression inExpression = new InExpression();` 用于创建一个 SQL `IN` 表达式。
- `inExpression.setLeftExpression(new Column(dimension.getBizName()));` 设置 `IN` 表达式的左侧字段名为维度的业务名。
- `inExpression.setRightExpression(expressionList);` 设置 `IN` 表达式的右侧为包含所有默认值的表达式列表。
- `expressions.add(inExpression);` 将这个 `IN` 表达式添加到表达式列表中。
- 如果 `MetricTable` 对象不为空，更新表中的维度信息。

8. **将新的表达式添加到 SQL 查询中**:
   - `sql = SqlAddHelper.addWhere(sql, expressions);` 使用 `SqlAddHelper` 工具类将表达式添加到 SQL 查询的 `WHERE` 子句中。
   - `queryStatement.getDataSetQueryParam().setSql(sql);` 更新查询语句中的 SQL 查询。

总结

- **整体流程**:
  1. 检查查询语句是否包含数据集查询参数和 SQL 查询字符串。
  2. 从语义模型中提取所有具有默认值的维度。
  3. 如果 SQL 查询中没有针对这些维度的过滤条件，将默认值作为过滤条件添加到 SQL 查询中。
  4. 更新查询语句中的 SQL 查询字符串。
- **用途**:
  - 该转换器在查询过程中自动处理维度的默认值，并将这些默认值添加到 SQL 查询的 `WHERE` 子句中。这对于确保查询的完整性和正确性非常重要，特别是在用户未指定某些维度的情况下，系统能够使用预定义的默认值来生成 SQL 查询。