Apache Calcite Table

Apache Calcite在Schema初始化后，获取Table实例，其中涉及到数据源数据的读取，Table的定义如下：



对于Csv Example来说用到的Table如下：



在SchemaFactory中创建Table如下：

*private Map<String, Table> createTableMap() {*

*final Source baseSource = Sources.of(directoryFile);*

*File[] files = directoryFile.listFiles( //获取表数据文件*

*new FilenameFilter() {*

*public boolean accept(File dir, String name) {*

*final String nameSansGz = trim(name, ".gz");*

*return nameSansGz.endsWith(".csv")*

*|| nameSansGz.endsWith(".json");*

*}*

*});*

*if (files == null) {*

*System.out.println("directory " + directoryFile + " not found");*

*files = new File[0];*

*}*

*// Build a map from table name to table; each file becomes a table.*

*final ImmutableMap.Builder<String, Table> builder = ImmutableMap.builder();*

*for (File file : files) { //根据文件创建对应表*

*Source source = Sources.of(file);*

*Source sourceSansGz = source.trim(".gz");*

*final Source sourceSansJson = sourceSansGz.trimOrNull(".json");*

*if (sourceSansJson != null) {*

*JsonTable table = new JsonTable(source);*

*builder.put(sourceSansJson.relative(baseSource).path(), table);*

*continue;*

*}*

*final Source sourceSansCsv = sourceSansGz.trim(".csv");*

*final Table table = createTable(source);*

*builder.put(sourceSansCsv.relative(baseSource).path(), table);*

*}*

*return builder.build();*

*}*

*/\*\*根据表flavor类型创建不同类型的表 \*/*

*private Table createTable(Source source) {*

*switch (flavor) {*

*case TRANSLATABLE:*

*return new CsvTranslatableTable(source, null);*

*case SCANNABLE:*

*return new CsvScannableTable(source, null);*

*case FILTERABLE:*

*return new CsvFilterableTable(source, null);*

*default:*

*throw new AssertionError("Unknown flavor " + this.flavor);*

*}*

*}*

在Table中定义了数据的读取方式，包括数据的读取、数据转换为列，其实现类为Enumerator，其相关接口如下：

* current，返回集合中的当前element
* moveNext，集合向后移动一位
* reset，重置
* close，释放enumerable使用的资源

其使用如下：

*public class CsvScannableTable extends CsvTable*

*implements ScannableTable {*

*CsvScannableTable(Source source, RelProtoDataType protoRowType) {*

*super(source, protoRowType);*

*}*

*public Enumerable<Object[]> scan(DataContext root) {*

*final int[] fields = CsvEnumerator.identityList(fieldTypes.size());*

*final AtomicBoolean cancelFlag = DataContext.Variable.CANCEL\_FLAG.get(root);*

*return new AbstractEnumerable<Object[]>() {*

*public Enumerator<Object[]> enumerator() {*

*return new CsvEnumerator<>(source, cancelFlag, false, null,*

*new CsvEnumerator.ArrayRowConverter(fieldTypes, fields));*

*}*

*};*

*}*

*}*

在CsvEnumerator中使用CSVReader从文件中读取数据，并完成数据的解析，如下所示：

*Class CsvEnumerator {*

*private final CSVReader reader;*

*private final String[] filterValues;*

*private final AtomicBoolean cancelFlag;*

*private final RowConverter<E> rowConverter;*

*}*

其核心方法moveNext的实现如下：

*public boolean moveNext() {*

*//从文件中读取数据*

*final String[] strings = reader.readNext();*

*//完成数据的列转换*

*current = rowConverter.convertRow(strings);*

*.....*

*}*

RowConverter的定义如下：

*static class ArrayRowConverter extends RowConverter<Object[]> {*

*private final CsvFieldType[] fieldTypes; //列类型*

*private final int[] fields;*

*//将String[] 根据数据库每行数据定义转换成行数据*

*public Object[] convertNormalRow(String[] strings) {*

*final Object[] objects = new Object[fields.length];*

*for (int i = 0; i < fields.length; i++) {*

*int field = fields[i];*

*objects[i] = convert(fieldTypes[field], strings[field]);*

*}*

*return objects;*

*}*

*.......*

*}*