Apache Calcite Schema

Apache Calcite通过JDBC执行数据的查询，其是应用程序和数据存储之间的数据处理引擎。Calcite并不管理数据，也没有标准的数据格式，访问数据需要编写一个适配器来告诉Calcite，将数据源中某种格式的数据集合视为Table来进行操作，其中适配器依赖Schema来运行：

* 使用Model文件来定义Schema
* SchemaFactory类创建Schema
* 在Schema中，定义了Table，每个Table知道如何扫描数据
* Calcite解析查询语句（SQL）并计划如何查询这些table，在查询时提醒表进行读取数据，来完成查询语句的执行

# Schema的定义

其中Model文件的内容示例如下：

*{*

*version: '1.0',*

*defaultSchema: 'SALES',*

*schemas: [*

*{*

*name: 'SALES',*

*type: 'custom',*

*factory: 'org.apache.calcite.adapter.csv.CsvSchemaFactory',*

*operand: {*

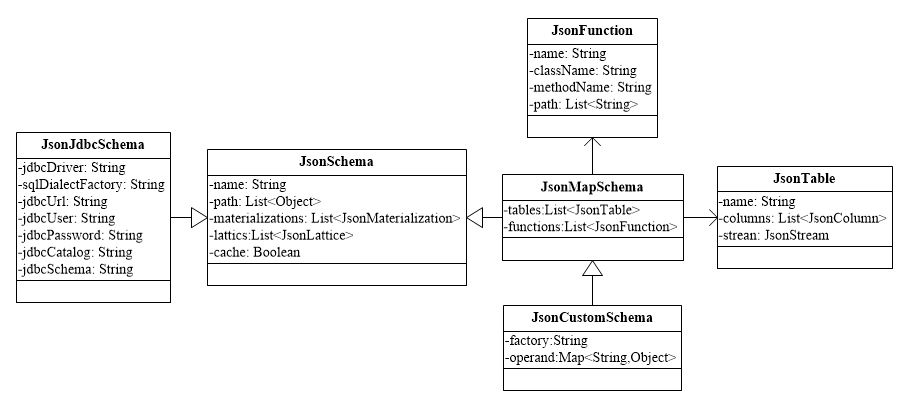
*directory: 'target/test-classes/sales'*

*}*

*}*

*]*

使用JsonCustomSchema类描述，类结构如下所示：



Calcite Models使用JSON格式文件来表示，也可以通过Schema SPI来构建（直接使用上述类，Java API初始化），下面介绍Model定义文件的Elements

1. root，根元素

*{*

*version: '1.0',*

*defaultSchema: 'mongo',*

*schemas: [Schema...]*

*}*

* version，版本号，配置为”1.0”
* defaultSchema，Model中定义的Schema名，Calcite使用的默认Schema

1. Schema，定义在root.schemas中，核心定义模块

*{*

*name: 'foodmart',*

*path: ['lib'],*

*type:[map,custom,jddc]*

*cache: true,*

*materializations: [Materialization...]*

*}*

* name, schema名称
* type，目前三种类型：Map Schema，Custom Schema和JDBC Schema，默认为Map
* path，定义SQL的路径，从其中解析Functions，配置为列表
* materializations，查询时进行物化的操作
* cache，默认true，是否缓存metadata，例如schema中生成的tables/functions/sub-schemas等

1. Map Schema，其定义格式如下：

*{*

*name: 'foodmart',*

*type: 'map',*

*tables: [Table...],*

*functions: [Function...]*

*}*

由于继承Schema，上述介绍的Element将省略

* tables，定义Table
* functions，定义Function

下面进行介绍

1. Custom Schema，其定义格式如下：

*{*

*name: 'mongo',*

*type: 'custom',*

*factory: 'org.apache.calcite.adapter.mongodb.MongoSchemaFactory',*

*operand: {*

*host: 'localhost',*

*database: 'test'*

*}*

*}*

* factory，Schema的Factory类，继承SchemaFactory
* operand，默认使用map，传给SchemaFactory的参数

1. JDBC Schema，格式如下：

*{*

*name: 'foodmart',*

*type: 'jdbc',*

*jdbcDriver: TODO,*

*jdbcUrl: TODO,*

*jdbcUser: TODO,*

*jdbcPassword: TODO,*

*jdbcCatalog: TODO,*

*jdbcSchema: TODO*

*}*

* jdbcDriver，Jdbc Driver类，不配置则由JDBC DriverManager自动选择
* jdbcUrl，默认String,Jdbc连接串，例如“jdbc:mysql://localhost/foodmart”
* jdbcUser，Jdbc 用户名
* jdbcPassword，Jdbc密码
* jdbcCatalog，JDBC数据源的Catalog
* jdbcSchema，JDBC数据源的初始Schema

1. table，root.schemas.tables，定义格式如下：

*{*

*name: 'sales\_fact',*

*columns: [Column...]*

*}*

* name，表名
* type，表类型，包括custom和view
* columns，列元素

1. View，类似于表，定义在root.schemas.tables中

*{*

*name: 'female\_emps',*

*type: 'view',*

*sql: "select \* from emps where gender = 'F'",*

*modifiable: true*

*}*

* Sql，SQL定义语句
* Path，解析查询的SQL路径
* Modifiable，View是否可变

8）Custom Table

*{*

*name: 'female\_emps',*

*type: 'custom',*

*factory: 'TODO',*

*operand: {*

*todo: 'TODO'*

*}*

*}*

* Factory，Table表的factory类，其实现TableFactory接口
* Operand，传给factory的参数

9）Stream，在表中定义是否为stream表，在root.schemas.tables.stream中定义，格式如下：

*{*

*stream: true,*

*history: false*

*}*

* stream，默认true，表是否允许Streaming
* history，默认false，stream的历史信息是否可用

1. Column，定义列名，配置在root.schemas.tables.columns中

*{ name: 'empno' }*

1. Function，定义在root.schemas.functions中，其定义格式如下：

*{*

*name: 'MY\_PLUS',*

*className: 'com.example.functions.MyPlusFunction',*

*methodName: 'apply',*

*path: []*

*}*

* name，函数名
* className，实现函数的类
* methodName，函数方法

其他如Lattice和Title等不再介绍，详细见链接：<http://calcite.apache.org/docs/model.html>

# Schema的使用

Apache Calcite基于定义的model文件，生成Schema，一般是通过SchemaFactory基于JsonSchema对象创建Schema，由ModuleHandler调用:

*public void visit(JsonCustomSchema jsonSchema) {*

*try {*

*final SchemaPlus parentSchema = currentMutableSchema("sub-schema");*

*checkRequiredAttributes(jsonSchema, "name", "factory");*

*final SchemaFactory schemaFactory =*

*AvaticaUtils.instantiatePlugin(SchemaFactory.class,*

*jsonSchema.factory);*

*final Schema schema =*

*schemaFactory.create(*

*parentSchema, jsonSchema.name, operandMap(jsonSchema, jsonSchema.operand));*

*final SchemaPlus schemaPlus = parentSchema.add(jsonSchema.name, schema);*

*populateSchema(jsonSchema, schemaPlus);*

*} catch (Exception e) {*

*throw new RuntimeException("Error instantiating " + jsonSchema, e);*

*}}*

以CsvSchema为例，其create源码如下：

*public Schema create(SchemaPlus parentSchema, String name,  
 Map<String, Object> operand) {  
 final String directory = (String) operand.get("directory");  
 .....*

*return new CsvSchema(directoryFile, flavor);  
}*

其operand为model.json中定义的参数，如：

*operand: {*

*directory: 'target/test-classes/sales'*

*}*

Apache Calcite的核心是Schema，其中描述了表结构，数据来源等信息，在Calcite中Schema的核心类如下所示：



针对每一种数据源都要实现相对的Schema，如MongoSchema、CassandraSchema、DruidSchema、SplunkSchema等，不再一一介绍。

客户端进行数据查询时，通过CalciteMetaImpl来调用Schema来进行查询，其类图如下：

