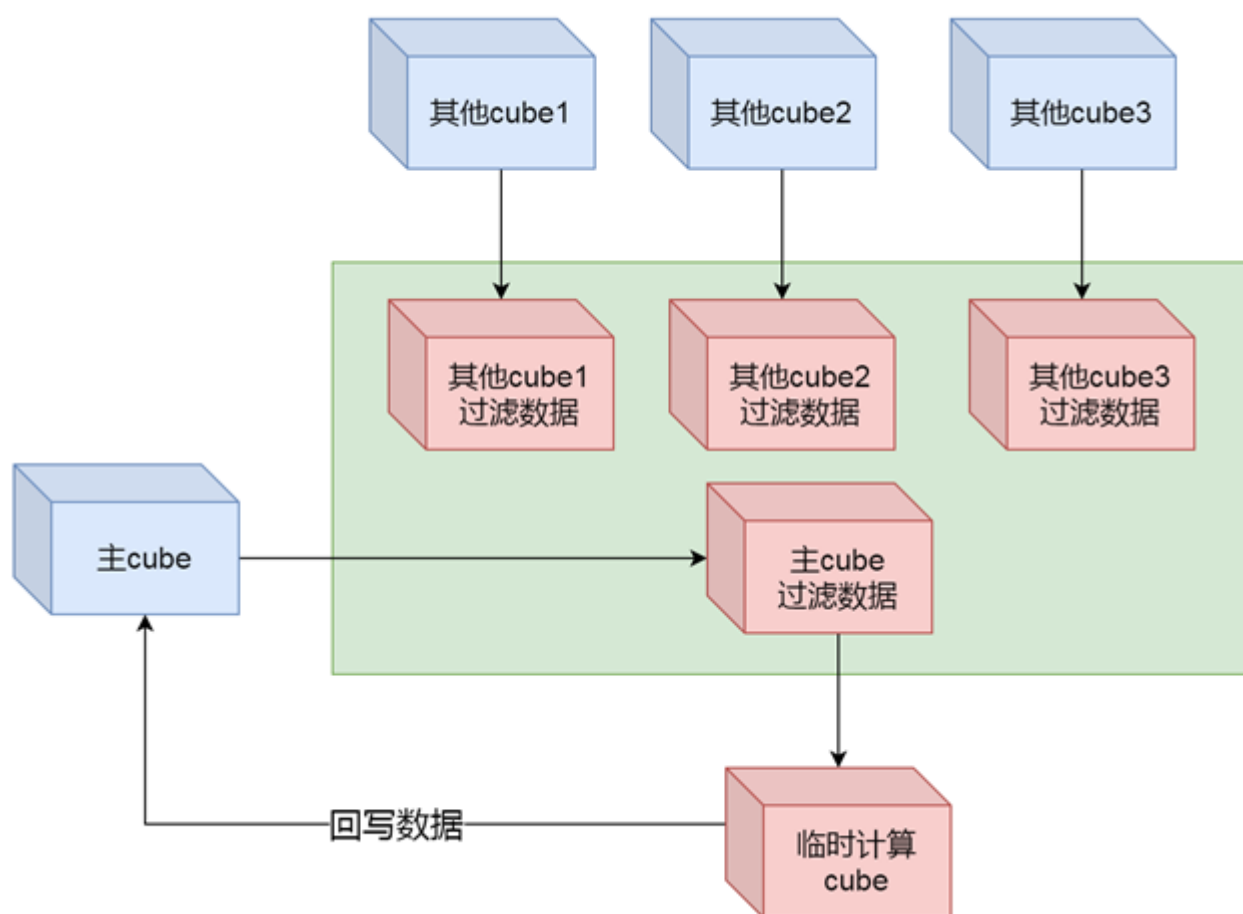


[返回首页](#)

## 跨cube计算

### 主要流程



现有的跨cube计算主要流程和实现的功能是：

实例内跨cube计算，在同一个多维库服务里，从其他cube中查询取数后，根据表达式将计算结果写回主cube；整个过程中，只会变更主cube的数据，跨cube数据仅仅是读取；

### 变更database属性

跨cube计算是基于存在业务关系的cube才能进行取数计算，所以需要业务指定可跨cube计算的范围权限，毫无关联的cube，不能进行取数计算，也没有任何业务语义。

```
static void alterCubeDatabase(OlapConnection olapConn, String cubeName, String database) {
    MetadataCommandInfo commandInfo = new MetadataCommandInfo();
    commandInfo.setMetadataType(MetadataTypes.Cube);
    commandInfo.setName(cubeName);
    commandInfo.setOwnerUniqueName(cubeName);
}
```

```

        commandInfo.setAction(CommandTypes.alter);
        commandInfo.getProperties().set("database", database);
        OlapCommand command = new OlapCommand(olapConn, commandInfo);
        command.executeNonQuery();
    }

    //将需要参与跨cube的database修改为db1
    demo.alterCubeDatabase(olapConn, "A_CUBE", "db1");
    demo.alterCubeDatabase(olapConn, "B_CUBE", "db1");
    demo.alterCubeDatabase(olapConn, "C_CUBE", "db1");

```

跨cube计算支持下面常见场景：

- \* 数据合并
- \* 数据计算
- \* 配置类计算

本文档跨cube计算的基础数据，有下面几个cube

mainTestCube  
mainTestCube2  
mainTestCube3  
exchangeCube

mainTestCube					
year	term	org	account	currency	m1
year0	term0	org0	account0	currency0	1.2
year0	term1	org0	account0	currency0	2.7
year0	term0	org1	account0	currency0	5
year0	term1	org1	account0	currency0	3.3

mainTestCube2					
year	term	org	account	currency	m1
year0	term0	org2	account0	currency0	21.2
year0	term1	org2	account0	currency0	22.7
year0	term0	org3	account0	currency0	35
year0	term1	org3	account0	currency1	33.3

mainTestCube3						
year	term	org	account	currency	otherdim	m1
year0	term0	org0	account1	currency0	otherdim0	1.2
year0	term1	org0	account1	currency0	otherdim0	2.7
year0	term0	org0	account2	currency0	otherdim0	5
year0	term1	org0	account2	currency0	otherdim1	3.3

exchangeCube			
year	org	currency	m1
year0	org0	currency0	6.5
year0	org1	currency0	1.1
year0	org0	currency1	10
year0	org1	currency1	3

数据合并、数据计算、配置类计算其实都是计算命令，下面是代码：

需要先创建mainTestCube对应的连接

```
OlapConnection olapConn = demo.getConnection("mainTestCube");
```

```
static void compute(OlapConnection olapConn) {
    FellLambdaExpressionItem lambda = new FellLambdaExpressionItem();
    lambda.setExpressLeft("org.org1");
    lambda.setExpression("mainTestCube2[org.org2]+mainTestCube2[org.org3]");

    ComputingCommandInfo computingCommandInfo = new ComputingCommandInfo();
    computingCommandInfo.getExpressionItems().add(lambda);
    computingCommandInfo.setMainDimName("org");
    computingCommandInfo.setMainMeaName("m1");

    OlapCommand cmd = new OlapCommand(olapConn, computingCommandInfo);
    cmd.executeCompute();
}
```

```
static void compute1(OlapConnection olapConn) {
    FellLambdaExpressionItem lambda = new FellLambdaExpressionItem();
    lambda.setExpressLeft("account.account0");
    lambda.setExpression("mainTestCube3[account.account1,otherdim.otherdim0]+
        mainTestCube3[account.account2,otherdim.otherdim0]");
}
```

```

ComputingCommandInfo computingCommandInfo = new ComputingCommandInfo();
computingCommandInfo.getExpressionItems().add(lambda);
computingCommandInfo.setMainMeaName("m1");
computingCommandInfo.addFilter("year", "year0", "year1");
computingCommandInfo.addFilter("term", "term0", "term1");

OlapCommand cmd = new OlapCommand(olapConn, computingCommandInfo);
cmd.executeCompute();
}

```

```

static void compute2(OlapConnection olapConn) {
    FellambdaExpressionItem lambda = new FellambdaExpressionItem();
    lambda.setExpressLeft("account.account0,currency.currency1");
    lambda.setExpression("[account.account0,currency.currency0] *
                        cube('exchangeTestCube','currency.currency1')");
    ComputingCommandInfo computingCommandInfo = new ComputingCommandInfo();
    computingCommandInfo.getExpressionItems().add(lambda);
    computingCommandInfo.setMainMeaName("m1");
    computingCommandInfo.addFilter("year", "year0", "year1");
    computingCommandInfo.addFilter("term", "term0", "term1");
    computingCommandInfo.addFilter("org", "org0", "org1");

    OlapCommand cmd = new OlapCommand(olapConn, computingCommandInfo);
    cmd.executeCompute();
}

```

## cube函数

表达式支持cube函数

```

公式1： cube(cubeName)
公式1： cube(cubeName,overrideKey)
公式2： cube(cubeName,overrideKey,joinMode)
公式4： cube(cubeName,sandboxName,measureName,expression)
公式5： cube(cubeName,sandboxName,measureName,expression,joinMode)

```

### 公式使用说明

跨cube 计算，操作的是 `cube` 时，可使用 `公式1`、`公式2`、`公式3`。当操作的是 `cube` 下的持久化沙箱时，只能使用 `公式4`、`公式5`

### 参数说明

**cubeName**：跨cube计算的cube名称，可为空，表示当前连接对应的cube

**overrideKey**：为维度成员键，与同cube相同，未指定的维度成员由当前同名的target带入，为空时全部维度都由target带入

**sandboxName**: 沙箱名称，可为空字符串（""），表示不操作沙箱，而是操作cube

**measureName:** 度量值名称，可为空字符串（"），表示使用计算命令上的 `MainMeaName` 的值

**joinMode：** 可缺省，默认值为normal

目前为以下值：

- normal
- hashjoin

可以指定hashjoin算法，如果一个cube指定一次hashjoin，则全部为hashjoin模式

使用cube函数替换以上代码：

```
static void compute(OlapConnection olapConn) {
    //lambda.setExpression("mainTestCube2[org.org2]+mainTestCube2[org.org3]");
    lambda.setExpression("cube('mainTestCube2','org.org2')+cube('mainTestCube2','org.org3')");
}
```

## 持久化沙箱跨cube计算

假设 mainTestCube2 中存在沙箱 s2, 上述操作变为取持久化沙箱 s2 的数据：

注意：操作持久化沙箱时，函数不能简写为 `cube[expression]`，只能使用 `cube` 函数

```
static void compute(OlapConnection olapConn) {

    lambda.setExpression("cube('mainTestCube2','s2','m1','org.org2')+cube('mainTestCube2','s2','m1',
'org.org3')");
}
```

## 临时沙箱跨cube计算

由于临时沙箱生命周期比较短，故只支持取其他 `cube` 数据，回写临时沙箱，不支持其他 `cube` 取临时沙箱数据进行计算。

```
static void compute(OlapConnection olapConn) {
    Closeable sandbox = olapConn.createSandbox();
    // 在该链接上执行的所有命令都是对沙箱的操作
    try {
        // 对沙箱执行各种操作，像以往代码一样进行保存，查询，计算或动态计算
        //lambda.setExpression("mainTestCube2[org.org2]+mainTestCube2[org.org3]");

        lambda.setExpression("cube('mainTestCube2','org.org2')+cube('mainTestCube2','org.org3')");
    } finally {
        try {
            // 沙箱使用完后，必须要手动关闭沙箱
            sandbox.close();
        } catch (IOException e) {
```

```

        e.printStackTrace();
    }
}
}

```

## hashJoin模式

hashJoin模式是特定的计算场景，较适用于类似算汇率，大cube包小cube时的乘法运算，不会产生多余的数据；

使用cube函数指定joinMode：

```

static void compute(OlapConnection olapConn) {
    lambda.setExpression("[year.year0]*cube('exchangeTestCube','year.year0','hashjoin')");
}

```

## 其他补充

### 跨cube名称

跨cube名称目前只能出现在表达式中，不能在filter里指定；

mainTestCube2只能出现在以下：

```

static void compute(OlapConnection olapConn) {
    //lambda.setExpression("mainTestCube2[org.org2]+mainTestCube2[org.org3]");
    lambda.setExpression("cube('mainTestCube2','org.org2')+cube('mainTestCube2','org.org3')");
}

```

## 计算表达式规范

1. 表达式左边指定了的维度，右边也必须指定；

year.year1,currency.currency0=mainTestCube2[year.year0]

左边出现了currency.currency0，右边也必须指定currency成员

2. 表达式中跨cube中多出来的维度必须指定；

account.account0=mainTestCube3[account.account1]+mainTestCube3[account.account2]

mainTestCube3中比主cube（mainTestCube）多出了维度otherdim，需要指定otherdim成员

## 跨cube成员不对齐问题

业务对于不对齐的维度成员现阶段要求交集后放到filter里，多维库这边处理是发现不对齐就直接抛异常；

这样能保证计算逻辑正确

处理逻辑：

cube1 b维度成员有 b1 b2 b3 b4

cube2 b维度成员有 b2 b3

cube3 b维度成员有 b3 b4

filter: b in b3

$\text{cube1}(a.a1) = \text{cube2}(a.a2) + \text{cube3}(a.a3)$

展开单元格计算过程是：

$\text{cube1}[a1,b3] = \text{cube2}[a2,b3] + \text{cube3}[a2,b3]$

上一篇：[有效维度组合规则](#)

下一篇：[tcp连接使用](#)