

[返回首页](#)

作者：董锦龙；时间：2023-04-20

功能的背景

业务使用报表功能，生成特殊报表的数据时，大部分情景下，特殊报表 == 月报，业务直接使用该规则生成即可，但是业务对2023年3月甲公司的特殊报想要自己填写，于是希望这个时候特殊报表 != 月报，是可以自行编辑的。

分段公式

分段公式介绍

```
a1
|__fun1= a2 + a3, 范围：白名单[b2],[b3] 黑名单[b3,c1], 优先级高 (priority=2)
|__fun2= a4 + a5, 范围：白名单[b3,c1] 黑名单[无], 优先级低 (priority=1)
|__fun3= a6, 范围：白名单[b4] 黑名单[b4,c2], 优先级低 (priority=0)
```

公式范围描述可参考 [全局公式范围](#)

案例公式表示为：

- 单元格[a1,b2,c1]中包含[b2,c1]，根据公式优先级，先判断是否满足fun1公式的范围，发现满足白名单且不再黑名单内，使用公式fun1，即[a1,b2,c1] = [a2,b2,c1] + [a3,b2,c1]
- 单元格[a1,b3,c1]中包含[b3,c1]，虽然满足公式fun1的白名单，但是在黑名单范围内，根据公式优先级继续向下找，满足公式fun2的白名单，即[a1,b3,c1] = [a4,b3,c1] + [a5,b3,c1]
- 单元格[a1,b4,c1]中包含[b4,c1]时，不满足fun1 和 fun2 的白名单，满足fun3的白名单且不再黑名，即[a1,b4,c1] = [a6,b4,c1]
- 单元格[a1,b4,c2]中包含[b4,c2]时，虽然满足公式fun3的白名单，但是在黑名单范围内，公式依旧不生效，[a1,b4,c2] 为存储类型单元格
- 单元格[a1,b5,c2]、[a1,b6,c2]、[a1,b4,c2] 等，公式均不生效，即都为存储类型单元格

分段公式创建

创建分段公式

```
c3
|__fun1= c1 + c2, 范围：scope_test_1, 优先级 (priority=0)
```

scope_test_1 表示为包含 b3 的单元格均生效，但是 [a1,b3] 和 [a2,b3] 除外

```
/**
 * 创建单个分段公式项
 */
public static PiecewisePairMetadataItem createSimplePiecewisePairItem() {
    // 创建公式有效范围
    BlackWhiteScopeItem scopeItem = new BlackWhiteScopeItem("scope_test_1");
```

```

scopeItem.getWhitelist().add(SliceScopeItem.create().add("b", "b3"));
scopeItem.getBlackList().add(DiscretePointScopeItem.create(new String[]{"a",
"b"}).add("a1,b3").add("a2,b3"));

// 返回分段公式项
return PiecewisePairMetadataItem.create("fun_1", 0, scopeItem)
.addFactor("c1", AggOperators.PLUS)
.addFactor("c2", AggOperators.PLUS);
}

public static void main(String[] args) {
    // 设置公式表达式 c3 = c1 + c2, 公式优先级为0, 引用范围 scope_test_1
    execPiecewisePairItem(CommandTypes.create,"c","c3", createSimplePiecewisePairItem());
}

/**
 * 操作分段公式
 *
 * @param items 分段公式项列表
 * @param type 可执行 create,alter,drop
 */
public static void execPiecewisePairItem(CommandTypes type,String dimension,String member,
PiecewisePairMetadataItem... items) {
    OlapConnection olapConn = OlapConnectionUtil.getConnection(cubeName);
    MetadataCommandInfo commandInfo = new MetadataCommandInfo();
    commandInfo.setMetadataType(MetadataTypes.PiecewiseExpression);
    commandInfo.setOwnerUniqueName(cubeName + "." + dimension + "." + member);
    commandInfo.setAction(type);
    Arrays.stream(items).forEach(item -> {
        commandInfo.getItems().add(item);
    });

    OlapCommand command = new OlapCommand(olapConn, commandInfo);
    command.executeNonQuery();
}

```

参数介绍：

1. name: 公式名称，在当前成员的分段公式组中具有唯一性。
2. priority: 公式优先级，数字类型，值越大优先级越高，不可为空。
3. scope：公式有效数据范围。

上面已经创建了分段公式：

```

c3
|__fun1= c1 + c2, 范围：scope_test_1, 优先级 (priority=0)

```

当发现公式不适用，需要在此基础上新增：

```

c3
|__fun2= c2, 范围：scope_test_2, 优先级 (priority=1)
|__fun1= c1 + c2, 范围：scope_test_1, 优先级 (priority=0)

```

`scope_test_2` 表示为包含 `b2` 的单元格均生效

直接再次创建即可：

```
public static PiecewisePairMetadataItem createPiecewisePairItem() {
    BlackWhiteScopeItem scopeItem = new BlackWhiteScopeItem("scope_test_2");
    scopeItem.getWhiteList().add(SliceScopeItem.create().add("b", "b2"));
    // 设置公式表达式 c3 = c2
    return PiecewisePairMetadataItem.create("fun_2", 1, scopeItem)
        .addFactor("c2", AggOperators.PLUS);
}
public static void main(String[] args) {
    execPiecewisePairItem(CommandTypes.create, "c", "c3", createPiecewisePairItem());
}
```

分段公式查询

通过函数API 接口方式进行查询：

1. 接口名称: `getFactorsV2`

2. 参数描述:

- cubeName: 当前查询的cube 的名称，必填
- path: 路径：表示获取当前cube的维度成员因子结构，path格式为：1、dimName 2、dimName.memberName 3、dimName@memberName

```
public static void findFactorsV2(String path) {
    OlapConnection olapConn = OlapConnectionUtil.getConnection(cubeName);
    PropertyBag propertyBag = new PropertyBag();
    propertyBag.set("cubeName", cubeName);
    propertyBag.set("path", path);
    FunctionCommandInfo commandInfo = new FunctionCommandInfo("getFactorsV2", propertyBag);
    OlapCommand command = new OlapCommand(olapConn, commandInfo);
    PropertyBag result = command.executeFunction();
    System.out.println(result.get("result"));
}
public static void main(String[] args) {
    findFactorsV2("c@c3");
}
```

得到的结果案例：

```
[{
  "name": "c3",
  "aggShieldRule": "all",
  "calcExpression": {
    "type": "piecewiseExpression.v1",
    "expressions": [{
      "calcExpression": {
        "type": "aggLambda",
        "factors": [{
```

```

        "operator": "+",
        "name": "c2"
    }]
},
"privateRange": {
    "type": "blackWhiteScope.v1",
    "blacks": [],
    "whites": [{
        "type": "sliceScope.v1",
        "slices": [{
            "dimension": "b",
            "members": ["b2"]
        }]
    }],
    "name": "scope_test_2"
},
"name": "fun_2",
"priority": 1
}, {
    "calcExpression": {
        "type": "aggLambda",
        "factors": [{
            "operator": "+",
            "name": "c1"
        }], {
            "operator": "+",
            "name": "c2"
        }
    }],
    "privateRange": {
        "type": "blackWhiteScope.v1",
        "blacks": [{
            "type": "sliceScope.v1",
            "slices": [{
                "dimension": "b",
                "members": ["b3"]
            }]
        }],
        "whites": [{
            "type": "discretePointScope.v1",
            "dimensions": ["a", "b"],
            "points": [
                ["a1", "b3"],
                ["a2", "b3"]
            ]
        }],
        "name": "scope_test_1"
    },
    "name": "fun_1",
    "priority": 0
}]
}
}]

```

3. 返回值描述：








- name：字符串类型，表示当前成员的名称
- storageType：字符串类型，表达式成员类型，目前存在以下三种表达式类型：存储、动态计算(dynamicCalc)、动态计算并存储(dynamicCalcAndStored)。如果没有该属性则是存储
- aggShieldRule：字符串类型，表示聚合屏蔽规则名称
- calcExpression：对象类型，表示当前成员的计算表达式
 - type：字符串类型，表示表达式类型，目前存在以下两种表达式类型：聚合表达式(aggLambda)、分段表达式(piecewiseExpression.v1)

各个表达式描述：

1. 分段表达式(piecewiseExpression)：

- expressions：对象类型数组，表示表达式项
 - name：字符串类型，表示当前分段公式项的名称
 - calcExpression：对象类型，表示当前表达式项的计算表达式
 - type：字符串类型，表示表达式类型，只允许存在聚合表达式(aggLambda)、空表达式(emptyExpression.v1)
 - priority：数字类型，表达式公式优先级，越大越优先
 - privateRange：对象类型，表示私有的公式数据范围，*目前只允许黑白名单范围对象*
 - globalRange：字符串类型，表示引用的全局的公式数据范围，*与privateRange 只能存在一个*

2. 聚合表达式(aggLambda)

- type：字符串类型，表示表达式类型
- factors：对象类型数组，表达包含的聚合因子列表
 - id：字符串类型，标识因子ID
 - name：字符串类型，标识因子名称
 - operator：字符串类型，标识因子运算符。目前存在运算符：、、、、、、

```
"calcExpression": {
  "type": "aggLambda",
  "factors": [{
    "name": "a2",
    "id": "0",
    "operator": "+"
  }, {
    "name": "a3",
    "id": "1",
    "operator": "+"
  }]
}
```

3. 空表达式(emptyExpression)

- type：字符串类型，表示表达式类型

```
"calcExpression": {
  "type": "emptyExpression.v1"
}
```

也可以通过多维库运维后台页面进行查询：

管理

testCube

追踪

管理

性能分析

慢语句查询

自定义方法

执行自定义方法

getFactorsV2

cubeName

testCube

path

c@c3

执行

result

```
1  [{
2    "name": "c3",
3    "aggShieldRule": "all",
4    "calcExpression": {
5      "type": "piecewiseExpression.v1",
6      "expressions": [{
7        "calcExpression": {
8          "type": "aggLambda",
9          "factors": [{
10             "operator": "+",
11             "name": "c2"
12           }]
13        },
14        "privateRange": {
15          "type": "blackWhiteScope.v1",
16          "blacks": [],
17          "whites": [{
18            "type": "sliceScope.v1",
19            "slices": [{
20              "dimension": "b",
21              "members": ["b2"]
22            }]
23          }]
24        }
25      }]
26    }
27  ]
```

删除分段公式

```
c3
|__fun2= c2 , 范围 : scope_test_2 , 优先级 ( priority=1 )
|__fun1= c1 + c2 , 范围 : scope_test_1 , 优先级 ( priority=0 )
```

希望删除fun1 公式：可通过定义的 公式项名称 进行删除

```
public static void main(String[] args) {
    execPiecewisePairItem(CommandTypes.drop, "c", "c3", new
    PiecewisePairMetadataItem("fun_1"));
}
```

修改分段公式

```
c3
|__fun2= c2, 范围: scope_test_2, 优先级 (priority=1)
```

修改为：

```
c3
|__fun2= c2 + c2 + c1, 范围: scope_test_2, 优先级 (priority=1)
```

可通过定义的 公式项名称 进行修改，如果 公式项名称 不存在修改报错。

```
public static void main(String[] args) {
    BlackWhiteScopeItem scopeItem = new BlackWhiteScopeItem("scope_test_2");
    scopeItem.getWhiteList().add(SliceScopeItem.create().add("b", "b2"));

    execPiecewisePairItem(CommandTypes.alter, "c", "c3",
    PiecewisePairMetadataItem.create("fun_2", 1, scopeItem)
    .addFactor("0", "c2", AggOperators.PLUS)
    .addFactor("1", "c2", AggOperators.PLUS)
    .addFactor("2", "c1", AggOperators.PLUS));
}
```

上述方式存在的一些弊端，分段公式中每一个公式项引用的范围是私有的范围，范围有更改的话，不得不修改当前这个公式项，这导致了频繁的元数据操作。

故上述案例只适合公式范围更改不频繁的场景，针对范围频繁变更的场景，可使用下面介绍的 全局公式有效范围。

引用全局的公式有效范围

全局公式范围描述可参考 [全局公式范围](#)

上面的 修改分段公式 例子我们发现 scope_test_2 这个范围被重复使用了。这个时候我们可以把他设置成全局的公式范围通过范围名称来引用。无需重复定义。

创建全局范围：

```
public static void main(String[] args) {
    execCubeDataScope(CommandTypes.create, createBlackWhiteScopeItem());
}
public static BlackWhiteScopeItem createBlackWhiteScopeItem() {
    BlackWhiteScopeItem scopeItem = new BlackWhiteScopeItem("scope_test_2");
}
```

```

        scopeItem.getWhitelList().add(SliceScopeItem.create().add("b", "b2"));
        return scopeItem;
    }

    /**
     * 操作数据范围
     *
     * @param scopeItems 具体的数据范围元数据信息列表
     * @param type 可执行 create,alter,drop,repair
     */
    public static void execCubeDataScope(CommandTypes type, CubeDataScopeItem... scopeItems) {
        OlapConnection olapConn = OlapConnectionUtil.getConnection(cubeName);
        MetadataCommandInfo commandInfo = new MetadataCommandInfo();
        commandInfo.setMetadataType(MetadataTypes.CubeDataScope);
        commandInfo.setOwnerUniqueName(cubeName); // cubeName
        commandInfo.setAction(type); // create,drop
        Arrays.stream(scopeItems).forEach(item -> {
            commandInfo.getItems().add(item);
        });

        OlapCommand command = new OlapCommand(olapConn, commandInfo);
        command.executeNonQuery();
    }
}

```

新增公式

```

c3
|__fun3= c2 + c1, 范围: scope_test_2, 优先级 (priority=2)
|__fun2= c2 + c2 + c1, 范围: scope_test_2, 优先级 (priority=1)

```

```

public static void main(String[] args) {
    execPiecewisePairItem(CommandTypes.create, "c", "c3",
        PiecewisePairMetadataItem.create("fun_3", 2, "scope_test_2")
            .addFactor("0", "c2", AggOperators.PLUS)
            .addFactor("1", "c1", AggOperators.PLUS));
}

```

PiecewisePairMetadataItem.create 是重载的方法，既可以传递PiecewisePairMetadataItem对象，也可以传递一个全局的公式范围字符串

通过多维库运维后台页面进行查询：

执行自定义方法

getFactorsV2

cubeName ?

testCube

path ?

c@c3

执行

result

```
1  [[
2    "name": "c3",
3    "aggShieldRule": "all",
4    "calcExpression": {
5      "type": "piecewiseExpression.v1",
6      "expressions": [{
7        "calcExpression": {
8          "type": "aggLambda",
9          "factors": [{
10             "operator": "+",
11             "id": "0",
12             "name": "c2"
13           }, {
14             "operator": "+",
15             "id": "1",
16             "name": "c1"
17           }
18         ],
19         "globalRange": "scope_test_2",
20         "name": "fun_3",
21         "priority": 2
22       }, {
23         "calcExpression": {
24           "type": "aggLambda"
```

全局范围和私有范围介绍：全局范围可以通过函数接口 `getCubeDataScopes` 查询，私有的范围与公式进行绑定。

基于成员命令进行分段公式操作

应用场景1：当创建一个成员的时候同时将分段公式也设置进去

```
c4
|__fun_1= c1, 范围：v_scope_test_1, 优先级（priority=1）
|__fun_2= c2, 范围：v_scope_test_2, 优先级（priority=2）
|__fun_3= c3, 范围：v_scope_test_3, 优先级（priority=3）
```

v_scope_test_1、v_scope_test_2、v_scope_test_3 均为全局范围 对应的白名单为：[a1]、[a2]、[a3]，均无黑名单

```

public static void createScope() {
    BlackWhiteScopeItem item1 = new BlackWhiteScopeItem("v_scope_test_1");
    item1.getWhiteList().add(SliceScopeItem.create()
        .add("a", "a1"));
    BlackWhiteScopeItem item2 = new BlackWhiteScopeItem("v_scope_test_2");
    item2.getWhiteList().add(SliceScopeItem.create()
        .add("a", "a2"));
    BlackWhiteScopeItem item3 = new BlackWhiteScopeItem("v_scope_test_3");
    item3.getWhiteList().add(SliceScopeItem.create()
        .add("a", "a3"));
    execCubeDataScope(CommandTypes.create, item1, item2, item3);
}

public static void execMember(CommandTypes type, String dimension, MemberMetadataItem item) {
    OlapConnection olapConn = OlapConnectionUtil.getConnection(cubeName);
    MetadataCommandInfo commandInfo = new MetadataCommandInfo();
    commandInfo.setMetadataType(MetadataTypes.Member);
    commandInfo.setOwnerUniqueName(cubeName + "." + dimension);
    commandInfo.setAction(type);
    commandInfo.getItems().add(item);
    OlapCommand command = new OlapCommand(olapConn, commandInfo);
    command.executeNonQuery();
}

public static ExpressionMetadataItem getExpressionMetadataItem() {
    PiecewiseExpressionMetadataItem item = new PiecewiseExpressionMetadataItem();
    List<PiecewisePairMetadataItem> piecewisePairs = item.getPiecewisePairs();
    piecewisePairs.add(PiecewisePairMetadataItem.create("fun_1", 1, "v_scope_test_1")
        .addFactor("c1", AggOperators.PLUS));
    piecewisePairs.add(PiecewisePairMetadataItem.create("fun_2", 2, "v_scope_test_2")
        .addFactor("c2", AggOperators.PLUS));
    piecewisePairs.add(PiecewisePairMetadataItem.create("fun_3", 3, "v_scope_test_3")
        .addFactor("c3", AggOperators.PLUS));
    return item;
}

public static void main(String[] args) {
    createScope();
    MemberMetadataItem item = new MemberMetadataItem("c4");
    item.setStorageType(MemberStorageTypes.DynamicCalc);
    item.setExpression(getExpressionMetadataItem());
    execMember(CommandTypes.create, "c", item);
}

```

应用场景2：当修改一个成员的时候同时将分段公式也设置进去

当前 c5 为聚合公式：

```

c5
|__c2
|__c3

```

```
[{
  "name": "c5",
  "storageType": "dynamicCalc",
  "aggShieldRule": "all",
  "calcExpression": {
    "type": "aggLambda",
    "factors": [{
      "name": "c2",
      "id": "0",
      "operator": "+"
    }, {
      "name": "c3",
      "id": "1",
      "operator": "+"
    }
  ]
}
]
```

修改为分段公式：

```
c5
|__fun_1= c1, 范围：v_scope_test_1, 优先级（priority=1）
|__fun_2= c2, 范围：v_scope_test_2, 优先级（priority=2）
|__fun_3= c3, 范围：v_scope_test_3, 优先级（priority=3）
```

```
public static void main(String[] args) {
    MemberMetadataItem item = new MemberMetadataItem("c5");
    item.setExpression(getExpressionMetadataItem());
    execMember(CommandTypes.alter, "c", item);
}
```

通过函数API 接口方式进行查询 `c@c5`，结果如下：

```
[{
  "name": "c5",
  "storageType": "dynamicCalc",
  "aggShieldRule": "all",
  "calcExpression": {
    "type": "piecewiseExpression.v1",
    "expressions": [{
      "calcExpression": {
        "type": "aggLambda",
        "factors": [{
          "operator": "+",
          "name": "c3"
        }
      ]
    },
    "globalRange": "v_scope_test_3",
    "name": "fun_3",
```

```

    "priority": 3
  }, {
    "calcExpression": {
      "type": "aggLambda",
      "factors": [{
        "operator": "+",
        "name": "c2"
      }]
    },
    "globalRange": "v_scope_test_2",
    "name": "fun_2",
    "priority": 2
  }, {
    "calcExpression": {
      "type": "aggLambda",
      "factors": [{
        "operator": "+",
        "name": "c1"
      }]
    },
    "globalRange": "v_scope_test_1",
    "name": "fun_1",
    "priority": 1
  }]
}
}]

```

发现改错了，希望修改回聚合公式：

```

public static void main(String[] args) {
    MemberMetadataItem item = new MemberMetadataItem("c5");
    List<AggFactorMetadataItem> list = new ArrayList<>();
    list.add(new AggFactorMetadataItem("0", "c2", AggOperators.PLUS));
    list.add(new AggFactorMetadataItem("1", "c3", AggOperators.PLUS));
    item.setFactors(list);
    execMember(CommandTypes.alter, "c", item);
}

```

不支持原有的方式修改（设置成员的修复因子）

```

C:\Java\jdk1.8.0_162\bin\java.exe ...
Exception in thread "main" kd.bos.olap.dataSources.OlapServerRuntimeException: 当前成员 c5 存在分段表达式信息，传递的因子无法处理，#MemberMetadataCommand1600 -
    at kd.bos.olap.shrek.dataSources.ShrekOlapCommandKt.validateResult(ShrekOlapCommand.kt:269)
    at kd.bos.olap.shrek.dataSources.ShrekOlapCommand.execute(ShrekOlapCommand.kt:240)
    at kd.bos.olap.shrek.dataSources.ShrekOlapCommand.updateMetadata(ShrekOlapCommand.kt:156)
    at kd.bos.olap.shrek.dataSources.ShrekOlapCommand.executeNonQuery(ShrekOlapCommand.kt:54)
    at kd.bos.olap.dataSources.OlapCommand.executeNonQuery(OlapCommand.kt:73)
    at olap.sample.demo.PiecewiseTest.execMember(PiecewiseTest.java:55)
    at olap.sample.demo.PiecewiseTest.main(PiecewiseTest.java:76)

```

可通过设置聚合表达式的方式进行修改：

```

public static void main(String[] args) {
    MemberMetadataItem item = new MemberMetadataItem("c5");
    item.setExpression(getAggExpressionMetadataItem());
    execMember(CommandTypes.alter, "c", item);
}
public static ExpressionMetadataItem getAggExpressionMetadataItem() {
    AggExpressionMetadataItem item = new AggExpressionMetadataItem();
    item.getFactors().add(new AggFactorMetadataItem("0", "c2", AggOperators.PLUS));
    item.getFactors().add(new AggFactorMetadataItem("1", "c3", AggOperators.PLUS));
    return item;
}

```

关于空表达式的用法

```

c5
|__fun_1= c1, 范围：v_scope_test_1, 优先级 (priority=1)
|__fun_2= c2, 范围：v_scope_test_2, 优先级 (priority=2)
|__fun_3= c3, 范围：v_scope_test_3, 优先级 (priority=3)

```

v_scope_test_1、v_scope_test_2、v_scope_test_3 均为全局范围 对应的白名单为：[a1]、[a2]、[a3]，均无黑名单

现在希望当遇到[a3,b2]的时候，公式不生效，很显然我们可以通过对fun_3 公式设置对应的和名单。但有些场景是不允许修改公式的。

比如基于时间维度考虑，某一年用fun_3公式，但是下一年必须得用fun_4公式

这个时候fun_4公式可以设置为空公式，公式范围 `v_scope_test_4` 白名单范围设置为[a3,b2]，表示满足该公式的单元格为 `存储单元格`

```

public static void main(String[] args) {
    execPiecewisePairItem(CommandTypes.create, "c", "c5", createEmptyPiecewisePairItem());
}

public static PiecewisePairMetadataItem createEmptyPiecewisePairItem() {
    // 创建公式有效范围
    BlackWhiteScopeItem scopeItem = new BlackWhiteScopeItem("v_scope_test_4");
    scopeItem.getWhiteList().add(DiscretePointScopeItem.create(new String[]{"a",
    "b"}).add("a3,b2"));

    // 返回分段公式项
    return PiecewisePairMetadataItem.create("fun_4", 4, scopeItem);
}

```

分段公式目前存在问题

假设cube 存在两个维度：时间维度、组织维度、度量值为利润

```

a1 两年总
|__a2 2020年
|__a3 2020年
b1 总公司
|__b2 甲组织
|__b3 乙组织

```

存在以下业务场景：2020年总公司的利润不是甲和乙组织的利润和，需要手动调整。2021年总公司的利润甲和乙组织的利润和，动态计算得来。公式设置：

```

b1
|__fun= b2 + b3 黑名单：a3

```

同时两年总计的乙组织的利润和不等2020年和20201年乙组织利润合计，需要手动调整。两年总计的甲公司利润等于2020年和20201年甲组织利润合计，动态计算得来。

公式设置：

```

a1
|__fun= a2 + a3 黑名单：b3

```

		2020	2021	两年总
		a3 -> [b]	a2	a1
甲	b3	4	8	12
乙	b2 -> [a]	2	9	*11
总公司	b1	*6	17	(计算结果有误)

由于当前实际业务的需求场景不会触发这种bug，暂时未处理。后续版本会变更计算架构规则，从而解决这个问题。

分段公式V2版本

```

a1
|__fun1= a1 + a2 + ... + a800000
|__fun2= a1 + a2 + ... + a800000 + a800001
|__fun3= a1 + a2 + ... + a800000 + a800001 + a800002

```

V1 版本中，如果 fun1、fun2 和 fun3 均存在80万左右的成员，且大部分成员均一致，导致总体元数据命令过大，不仅传输量更大且执行也慢。

针对这种场景，可以将公共的因子提取出来。故可使用 分段公式V2 版进行改造。

创建成员时设置分段公式V2

假设需要在 c4 成员上面设置分段公式，公式信息如下：

```

c4
|__fun1= c1 + c2 + c3, 范围: null, 优先级 (priority=1)
|__fun2= c1 + c2 + c3 + c6 + c7 + c8, 范围: scope_test_2, 优先级 (priority=100)
|__fun3= c1 + c2 + c3 + c9 + c10, 范围: scope_test_3, 优先级 (priority=101)

```

提取出相同的因子到 `base` 表达式中, 结果如下:

```

c4
|__base= c1 + c2 + c3
|__fun1= 空, 范围: null, 优先级 (priority=1)
|__fun2= c6 + c7 + c8, 范围: scope_test_2, 优先级 (priority=100)
|__fun3= c9 + c10, 范围: scope_test_3, 优先级 (priority=101)

```

如果 `c4` 成员尚未创建, 可在成员创建时进行公式初始化, 具体代码实现如下:

```

/**
 * 创建分段公式V2版本
 */
public static PiecewiseExpressionMetadataItemV2 createPiecewiseExpressionV2() {
    PiecewiseExpressionMetadataItemV2 expression = new PiecewiseExpressionMetadataItemV2();
    // 设置base 表达式
    AggExpressionMetadataItem base = new AggExpressionMetadataItem();
    List<AggFactorMetadataItem> factors = base.getFactors();
    factors.add(new AggFactorMetadataItem("c1", AggOperators.PLUS));
    factors.add(new AggFactorMetadataItem("c2", AggOperators.PLUS));
    factors.add(new AggFactorMetadataItem("c3", AggOperators.PLUS));
    expression.setBaseExpression(base);

    // 设置fun1 优先级最低, 兜底公式, 不设置范围, 高优先级不匹配公式的时候, 一定会匹配到兜底公式
    AggExpressionMetadataItem fun1 = new AggExpressionMetadataItem();
    PiecewisePairMetadataItem fun1Pair = new PiecewisePairMetadataItem("fun1");
    fun1Pair.setCalcExpression(fun1);
    fun1Pair.setPriority(1);
    expression.getPiecewisePairs().add(fun1Pair);

    // 设置fun2
    AggExpressionMetadataItem fun2 = new AggExpressionMetadataItem();
    PiecewisePairMetadataItem fun2Pair = new PiecewisePairMetadataItem("fun2");
    fun2Pair.setCalcExpression(fun2);
    fun2Pair.setPriority(100);
    // 设置范围
    BlackWhiteScopeItem scopeItem = new BlackWhiteScopeItem("scope_test_2");
    scopeItem.getWhiteList().add(SliceScopeItem.create().add("b", "b2"));
    fun2Pair.setScope(scopeItem);

    // 设置表达式
    List<AggFactorMetadataItem> factors2 = fun2.getFactors();
    factors2.add(new AggFactorMetadataItem("c6", AggOperators.PLUS));
    factors2.add(new AggFactorMetadataItem("c7", AggOperators.PLUS));
    factors2.add(new AggFactorMetadataItem("c8", AggOperators.PLUS));
    expression.getPiecewisePairs().add(fun2Pair);
}

```

```

// 设置fun3
AggExpressionMetadataItem fun3 = new AggExpressionMetadataItem();
PiecewisePairMetadataItem fun3Pair = new PiecewisePairMetadataItem("fun3");
fun3Pair.setCalcExpression(fun3);
fun3Pair.setPriority(101);
// 设置范围
BlackWhiteScopeItem scopeItem3 = new BlackWhiteScopeItem("scope_test_3");
scopeItem3.getWhiteList().add(SliceScopeItem.create().add("b", "b3"));
fun3Pair.setScope(scopeItem3);

// 设置表达式
List<AggFactorMetadataItem> factors3 = fun3.getFactors();
factors3.add(new AggFactorMetadataItem("c9", AggOperators.PLUS));
factors3.add(new AggFactorMetadataItem("c10", AggOperators.PLUS));
expression.getPiecewisePairs().add(fun3Pair);

return expression;
}

public static void main(String[] args) {
    MemberMetadataItem item = new MemberMetadataItem("c4");
    item.setStorageType(MemberStorageTypes.DynamicCalc);
    item.setExpression(createPiecewiseExpressionV2());
    execMember(CommandTypes.create, "c", item);
}

```

注意：目前分段公式V2版本必须有一个范围为 `null` 的兜底公式，即优先级最低的公式范围必须是 `null`。

修改成员时设置分段公式

如果 `c4` 成员尚已经创建，可在修改成员时进行公式创建，具体代码实现如下：

```

public static void main(String[] args) {
    MemberMetadataItem item = new MemberMetadataItem("c4");
    item.setStorageType(MemberStorageTypes.DynamicCalc);
    item.setExpression(createPiecewiseExpressionV2());
    execMember(CommandTypes.alter, "c", item);
}

```

增加一个公式项

当 `c4` 成员已经存在了分段公式V2，具体公式内容如下：

```

c4
|__base= c1 + c2 + c3
|__fun1= 空，范围：null，优先级（priority=1）
|__fun2= c6 + c7 + c8，范围：scope_test_2，优先级（priority=100）
|__fun3= c9 + c10，范围：scope_test_3，优先级（priority=101）

```

此时希望增加 `fun4 = c11 + c12+ c13`，可执行分段公式项创建，具体代码实现如下：

```

/**
 * 操作分段公式
 *
 * @param items 分段公式项列表
 * @param type 可执行 create,alter,drop
 */
public static void execPiecewisePairItem(CommandTypes type, String dimension, String member,
PiecewisePairMetadataItem... items) {
    OlapConnection olapConn = OlapConnectionUtil.getConnection(cubeName);
    MetadataCommandInfo commandInfo = new MetadataCommandInfo();
    commandInfo.setMetadataType(MetadataTypes.PiecewiseExpression);
    commandInfo.setOwnerUniqueName(cubeName + "." + dimension + "." + member);
    commandInfo.setAction(type);
    Arrays.stream(items).forEach(item -> {
        commandInfo.getItems().add(item);
    });

    OlapCommand command = new OlapCommand(olapConn, commandInfo);
    command.executeNonQuery();
}

public static PiecewisePairMetadataItem createFun4() {
    BlackWhiteScopeItem scopeItem = new BlackWhiteScopeItem("scope_test_4");
    scopeItem.getWhiteList().add(SliceScopeItem.create().add("b", "b4"));
    // 设置公式表达式 c3 = c2
    return PiecewisePairMetadataItem.create("fun4", 102, scopeItem)
        .addFactor("c11", AggOperators.PLUS)
        .addFactor("c12", AggOperators.PLUS)
        .addFactor("c13", AggOperators.PLUS);
}

public static void main(String[] args) {
    execPiecewisePairItem(CommandTypes.create, "c", "c4", createFun4());
}

```

修改一个公式项

此时希望增加 `fun4 = c11 + c12+ c13 + c14` , 可执行分段公式项修改, 具体代码实现如下:

```

public static PiecewisePairMetadataItem createUpdFun4() {
    BlackWhiteScopeItem scopeItem = new BlackWhiteScopeItem("scope_test_4");
    scopeItem.getWhiteList().add(SliceScopeItem.create().add("b", "b4"));
    // 设置公式表达式 c3 = c2
    return PiecewisePairMetadataItem.create("fun4", 102, scopeItem)
        .addFactor("c11", AggOperators.PLUS)
        .addFactor("c12", AggOperators.PLUS)
        .addFactor("c13", AggOperators.PLUS)
        .addFactor("c14", AggOperators.PLUS);
}

```

```
public static void main(String[] args) {
    execPiecewisePairItem(CommandTypes.alter, "c", "c4", createUpdFun4());
}
```

删除一个公式项

此时希望将 `fun2` 分段公式项删除，具体代码实现如下：

```
public static void main(String[] args) {
    execPiecewisePairItem(CommandTypes.drop, "c", "c4", new PiecewisePairMetadataItem("fun2"));
}
```

base 表达式中新增因子

当 `c4` 成员已经存在了分段公式V2，具体公式内容如下：

```
c4
|__base= c1 + c2 + c3
|__fun1= c6 + c9, 范围：null, 优先级 (priority=1)
|__fun2= c6 + c7 + c8, 范围：scope_test_2, 优先级 (priority=100)
|__fun3= c9 + c10, 范围：scope_test_3, 优先级 (priority=101)
```

此时希望 `base` 公式中增加因子 `c11` 和 `c12`，具体代码实现如下：

```
/**
 * 成员因子操作
 *
 * @param type      create、drop、alter、repair
 * @param dimension 维度
 * @param member    成员
 * @param items     因子列表
 */
public static void execAggFactor(CommandTypes type, String dimension, String member,
AggFactorMetadataItem... items) {
    OlapConnection olapConn = OlapConnectionUtil.getConnection(cubeName);
    MetadataCommandInfo commandInfo = new MetadataCommandInfo();
    commandInfo.setMetadataType(MetadataTypes.AggFactor);
    commandInfo.setOwnerUniqueName(cubeName + "." + dimension + "." + member);
    commandInfo.setAction(type);
    Arrays.stream(items).forEach(item ->
        commandInfo.getItems().add(item)
    );

    OlapCommand command = new OlapCommand(olapConn, commandInfo);
    command.executeNonQuery();
}

public static void main(String[] args) {
    execAggFactor(CommandTypes.create, "c", "c4",
        new AggFactorMetadataItem("c11", AggOperators.PLUS),
        new AggFactorMetadataItem("c12", AggOperators.PLUS));
}
```

```
}
```

base 表达式中删除因子

此时希望 `base` 公式中删除因子 `c2` 和 `c3`，具体代码实现如下：

```
public static void main(String[] args) {  
    execAggFactor(CommandTypes.drop, "c", "c4",  
        new AggFactorMetadataItem("c2", AggOperators.PLUS),  
        new AggFactorMetadataItem("c3", AggOperators.PLUS));  
}
```

base 表达式中修复因子

如果希望将 `base` 中的因子全部删除，再以新的因子清单为准，可执行 `repair` 修复操作。

比如现有 `base = c1 + c11 + c12` 改为 `base = c2 + c3`

```
public static void main(String[] args) {  
    execAggFactor(CommandTypes.repair, "c", "c4",  
        new AggFactorMetadataItem("c2", AggOperators.PLUS),  
        new AggFactorMetadataItem("c3", AggOperators.PLUS));  
}
```