

[返回首页](#)

作者：董锦龙；时间：2023-04-20

功能的背景

全局公式数据范围主要是提供给分段公式的有效范围范围进行引用。当多个分段公式引用同一个范围对象时可达到复用的效果。

法人组织 是一个组织的范围，定义一个通用全局范围命名，将凡是法人的组织选出来，在作 **科目统计** 时候凡是法人的范围即可直接引用这个全局范围

公式数据范围

目前支持公式数据范围类型：

1. 散点：点阵的方式描述，可以精确到某个具体的单元格。
2. 切片：通过多维度成员之间的笛卡尔积描述具体的范围。
3. 黑白名单：内置黑、白名单范围，黑白名单范围其实是由一个个散点和切片范围组成。

关于黑白名单的补充描述：以招聘为例，当应聘者进入公司时，保安手里拿着一份应聘白名单，确认这个人是来公司应聘的，故放行。但是在HR 手里有一个行业人员黑名单，当面试者进来时，HR发现他在黑名单里，故该应聘人员还是无法进行面试。

首先符合白名单，且不在黑名单范围，公式生效。

应用具体需求场景：业务的情景维度存在 **特殊报表** 和 **月报**。默认情况下 **特殊报表** = **月报**，但是有一些 **特殊的情况**，希望公式不生效，该 **特殊的情况** 可用黑名单范围表示。

目前公式只能引用黑白名单的数据范围

多维库V8.9.0 版本暂时只支持公式上挂载黑白名单范围。

全局公式数据范围操作

假设cube 存在三个维度：组织、科目、币别。分表表示为a、b、c。

科目 成员如下：

```
a1 总公司
|__a2 甲组织
|__a3 乙组织
```

组织 成员如下：

```
b1 资产
|__b2 现金
|__b3 银行存款
```

币别 成员如下：

c1 人民币
c2 美元
c3 欧元

创建散点数据范围对象

要求：创建一个包含 [甲组织,现金]、[乙组织,现金] 的范围

```
public static DiscretePointScopeItem createDiscretePointScopeItem() {
    return DiscretePointScopeItem.create(new String[]{"a","b"})
        .add("a2,b3")
        .add("a3,b3");
}
```

表示的范围区间：

[a2,b3] [a3,b3]

范例单元格	是否命中
[a2,b3,c1]	true
[a3,b3,c2]	true
[a2,b2,c1]	false
[a3,b2,c2]	false

创建切片数据范围对象

要求：创建一个包含 [甲组织,现金]、[乙组织,现金]、[甲组织,银行存款]、[乙组织,银行存款] 的范围

```
public static SliceScopeItem createSliceScopeItem() {
    /**
     * 得到的数据范围为笛卡尔积： [a2,b2] [a2,b3] [a3,b2] [a3,b3]
     */
    return SliceScopeItem.create().add("a","a2","a3").add("b","b2","b3");
}
```

表示的范围区间：

[a2,b2] [a2,b3] [a3,b2] [a3,b3]

范例单元格	是否命中
[a2,b3,c1]	true
[a3,b3,c2]	true
[a2,b2,c1]	true
[a3,b2,c2]	true

创建黑白名单公式数据范围对象

黑白名单范围：

- 1. 白名单： [甲组织, 现金]、[乙组织, 现金]、[甲组织, 银行存款]、[乙组织, 银行存款]
- 2. 黑名单： [甲组织, 现金]、[乙组织, 现金]

```
public static BlackWhiteScopeItem createBlackWhiteScopeItem() {
    BlackWhiteScopeItem item = new BlackWhiteScopeItem("test_scope_1");
    item.getBlackList().add(createSliceScopeItem());
    item.getWhitelist().add(createDiscretePointScopeItem());
    return item;
}
```

公式范围元数据操作方法

```
/**
 * 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.scopeName.blacks.point_name
    commandInfo.setAction(type); // create,drop
    Arrays.stream(scopeItems).forEach(item -> {
        commandInfo.getItems().add(item);
    });

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

创建公式数据范围

```
public static void main(String[] args) {
    execCubeDataScope(CommandTypes.create, createBlackWhiteScopeItem());
}
```

如何验证是否创建成功？

公式数据范围查询

通过函数API 接口方式进行查询：

1. 接口名称: `getCubeDataScopes`
2. 参数描述:
 - cubeName: 当前查询的cube 的名称，必填
 - name: 当前全局公式范围的名称，非必填。为空表示查询cube下所有的公式范围

```
public static void findCubeDataScope(String scopeName) {
    OlapConnection olapConn = OlapConnectionUtil.getConnection(cubeName);
    PropertyBag propertyBag = new PropertyBag();
    propertyBag.set("cubeName", cubeName);
    if(scopeName != null){
        propertyBag.set("name", scopeName);
    }
    FunctionCommandInfo commandInfo = new FunctionCommandInfo("getCubeDataScopes", propertyBag);
    OlapCommand command = new OlapCommand(olapConn, commandInfo);
    PropertyBag result = command.executeFunction();
    System.out.println(result.get("result"));
}
```

得到的结果案例：

```
[{
  "type": "blackWhiteScope.v1",
  "blacks": [{
    "type": "sliceScope.v1",
    "slices": [{
      "dimension": "a",
      "members": ["a2", "a3"]
    }, {
      "dimension": "b",
      "members": ["b2", "b3"]
    }]
  }],
  "whites": [{
    "type": "discretePointScope.v1",
    "dimensions": ["a", "b"],
    "points": [
      ["a2", "b3"],
      ["a3", "b3"]
    ]
  }],
}]
```

```
"name": "test_scope_1"  
}]
```

3. 返回值描述：

- name：字符串类型，表示当前范围的名称
- type：字符串类型，表示范围类型，目前存在以下三种：黑白名单(blackWhiteScope.v1)、切片(sliceScope.v1)、散点(discretePointScope.v1)
- 黑白名单范围：
 - whites：对象数组类型，表示白名单列表，内部可以是多个范围对象
 - blacks：对象数组类型，表示黑名单列表，内部可以是多个范围对象
- 切片范围：
 - slices：对象数组类型，表示一个切片，由多个维度和成员的笛卡尔积形成
 - dimension：字符串类型，表示维度名称
 - members：字符串数组类型，表示成员集合
- 散点范围：
 - dimensions：字符串数组类型，表示维度列表
 - points：字符串数组类型，表示具体的点信息

也可以通过多维库运维后台页面进行查询：

执行自定义方法

getCubeDataScopes

cubeName ?

testCube

name ?

name

执行

result

```
1  [{
2    "type": "blackWhiteScope.v1",
3    "blacks": [{
4      "type": "sliceScope.v1",
5      "slices": [{
6        "dimension": "a",
7        "members": ["a2", "a3"]
8      }, {
9        "dimension": "b",
10       "members": ["b2", "b3"]
11     }]
12   }],
13   "whites": [{
14     "type": "discretePointScope.v1",
15     "dimensions": ["a", "b"],
16     "points": [
17       ["a2", "b3"],
18       ["a3", "b3"]
19     ]
20   }],
21   "name": "test_scope_1"
22 }]
```

修改公式数据范围

当发现范围规则不适用时，可根据 范围规则名称 进行修改。如果有分段公式引用了这个范围，会同步更新。

黑白名单范围：

1. 白名单： [甲组织, 现金]、[乙组织, 现金]、[甲组织, 银行存款]
2. 黑名单： [甲组织, 现金, 人民币]

```

public static void main(String[] args) {
    BlackWhiteScopeItem item = new BlackWhiteScopeItem("test_scope_1");
    item.getWhiteList().add(DiscretePointScopeItem.create(new String[]{"a", "b"}))
        .add("a2,b2")
        .add("a3,b2")
        .add("a2,b3"));
    item.getWhiteList().add((DiscretePointScopeItem.create(new String[]{"a", "b",
    "c"}).add("a2,b2,c1"))));
    execCubeDataScope(CommandTypes.alter, item);
}

```

修复公式数据范围

修复与修改的区别：假设原公式范围有4个：test_scope_1、test_scope_2、test_scope_3、test_scope_4

修改：传入对应的公式范围对象，根据该对象的范围名称进行判断，如果公式范围名称存在，则进行对应的修改，如果名称不存在则报错。

比如传入范围名称为 test_scope_1、test_scope_2 的对象能完成修改，但是传入名称为 test_scope_5、test_scope_6 的对象则直接报错。

修改后的范围还是：test_scope_1、test_scope_2、test_scope_3、test_scope_4

修复：传入需要修复的公式范围对象，如果对象的范围名称存在，则为修改，不存在则新增。原公式范围不在此次修复范围内的则删除。

比如传入范围名称为 test_scope_1、test_scope_2、test_scope_5、test_scope_6 的对象，test_scope_1、test_scope_2 的对象能完成修改。test_scope_5、test_scope_6 的对象则新增。原有 test_scope_3、test_scope_4 则删除

修复后的范围：test_scope_1、test_scope_2、test_scope_5、test_scope_6

案例：去掉之前已存在的公式范围 test_scope_1，并新建一个新的范围 test_scope_1_new，其只包含白名单，不包含黑名单。

test_scope_1_new 白名单范围：[甲组织,现金]、[乙组织,现金]、[甲组织,银行存款]、[乙组织,银行存款]

```

public static void main(String[] args) {
    BlackWhiteScopeItem item = new BlackWhiteScopeItem("test_scope_1_new");
    item.getWhiteList().add(SliceScopeItem.create()
        .add("a", "a2", "a3")
        .add("b", "b2", "b3"));

    execCubeDataScope(CommandTypes.repair, item);
}

```

删除公式数据范围

可以根据 范围规则名称 进行范围删除。但如果该范围被公式引用了，则不能删除。需要先删除公式，再删除范围

```
public static void main(String[] args) {
    execCubeDataScope(CommandTypes.drop, new BlackWhiteScopeItem("test_scope_1_new"));
}
```

针对黑白名单中的散点数据范围进行新增和删除操作

该功能仅支持对 全局范围 的 黑白名单 中的 散点范围 进行操作。注意 散点范围 也必须给一个唯一的名称

背景：由于目前对公式数据范围进行操作的元数据命令中，不支持对 某个范围内部的一些信息 做修改，只能 整个范围进行覆盖修改 操作。实际应用中造成使用不方便。

案例：在全局范围内定义一个包含散点的黑白名单，由于一开始不知道散点的内容，允许散点内容为空，*但是必须要声明维度信息*

```
public static void main(String[] args) {
    BlackWhiteScopeItem item = new BlackWhiteScopeItem("my_scope");
    item.getBlackList().add(DiscretePointScopeItem.create("my_scope_black_01", new String[]{"a", "b"}));
    execCubeDataScope(CommandTypes.create, item);
}
```

新增操作

全局黑白名单范围 my_scope 的黑名单列表中包含一个散点 my_scope_black_01，维度为 [a,b]，内容为空

```
[{
  "type": "blackWhiteScope.v1",
  "blacks": [{
    "type": "discretePointScope.v1",
    "dimensions": ["a", "b"],
    "name": "my_scope_black_01",
    "points": []
  }],
  "whites": [],
  "name": "my_scope"
}]
```

此时希望在黑白名单范围 my_scope 的黑名单列表的散点 my_scope_black_01 中新增一个范围数据 [a1,b1], [a2,b2]

```
public static void main(String[] args) {
    String blackWhiteScopeName = "my_scope";
    String discretePointScopeName = "my_scope_black_01";
    String ownerName = cubeName + "." + blackWhiteScopeName + "." + "blacks" + "." + discretePointScopeName;
    addPointScope(ownerName, DiscretePointScopeItem.create(new String[]{"a", "b"}).add("a1,b1").add("a2,b2"));
}

/**
```



```

* 增加一个散点范围中的点信息
*
* @param ownerName 拥有者名称：cubeName.scopeName.blacks.discretePointScopeName
*                或者 cubeName.scopeName.whites.discretePointScopeName
* @param scopeItem 新增的散点信息
*/
public static void addPointScope(String ownerName, DiscretePointScopeItem scopeItem) {
    OlapConnection olapConn = OlapConnectionUtil.getConnection(cubeName);
    MetadataCommandInfo commandInfo = new MetadataCommandInfo();
    commandInfo.setMetadataType(MetadataTypes.CubeDataScope);
    commandInfo.setOwnerUniqueName(ownerName);
    commandInfo.setAction(CommandTypes.create);
    commandInfo.getItems().add(scopeItem);
    OlapCommand command = new OlapCommand(olapConn, commandInfo);
    command.executeNonQuery();
}

```

关于 `ownerName` 拥用者名称介绍：

cube名称.全局黑白名单范围名称.黑名单列表.黑名单列表下的散点范围名称
 或者
 cube名称.全局黑白名单范围名称.白名单列表.白名单列表下的散点范围名称

其中 黑名单列表 和 白名单列表 为固定字符串：`blacks` 和 `whites`

修改后的范围

```

[{
  "type": "blackWhiteScope.v1",
  "blacks": [{
    "type": "discretePointScope.v1",
    "dimensions": ["a", "b"],
    "name": "my_scope_black_01",
    "points": [
      ["a1", "b1"],
      ["a2", "b2"]
    ]
  }],
  "whites": [],
  "name": "my_scope"
}]

```

删除操作

此时希望在黑白名单范围 `my_scope` 的黑名单列表的散点 `my_scope_black_01` 中删除一个范围数据 `[a1,b1]`

```

public static void main(String[] args) {
    String blackWhiteScopeName = "my_scope";
    String discretePointScopeName = "my_scope_black_01";
    String ownerName = cubeName + "." + blackWhiteScopeName + "." + "blacks" + "." +
discretePointScopeName;
}

```

```

        dropPointScope(ownerName, DiscretePointScopeItem.create(new String[]{"a",
"b"}).add("a1,b1"));
    }

/**
 * 删除一个散点范围中的点信息
 *
 * @param ownerName 拥有者名称：cubeName.scopeName.blacks.discretePointScopeName
 *                  或者 cubeName.scopeName.whites.discretePointScopeName
 * @param scopeItem 删除的散点信息
 */
public static void dropPointScope(String ownerName, DiscretePointScopeItem scopeItem) {
    OlapConnection olapConn = OlapConnectionUtil.getConnection(cubeName);
    MetadataCommandInfo commandInfo = new MetadataCommandInfo();
    commandInfo.setMetadataType(MetadataTypes.CubeDataScope);
    commandInfo.setOwnerUniqueName(ownerName);
    commandInfo.setAction(CommandTypes.drop);
    commandInfo.getItems().add(scopeItem);
    OlapCommand command = new OlapCommand(olapConn, commandInfo);
    command.executeNonQuery();
}

```

修改后的范围：

```

[ {
  "type": "blackWhiteScope.v1",
  "blacks": [ {
    "type": "discretePointScope.v1",
    "dimensions": ["a", "b"],
    "name": "my_scope_black_01",
    "points": [
      ["a2", "b2"]
    ]
  } ],
  "whites": [],
  "name": "my_scope"
} ]

```

删除散点支持通配符 *

当前新增不支持通配符。与底层存储有关，存储的是具体的一个个点，容易对用户造成歧义。

应用场景：散点存在三个维度，`年`，`期间`，`组织`，用户希望一次性将某个年和期间内存在的所有散点信息全部删除掉。

案例：将散点中包含 `a2` 的点全部删除。

```
public static void main(String[] args) {
    String blackWhiteScopeName = "my_scope";
    String discretePointScopeName = "my_scope_black_01";
    String ownerName = cubeName + "." + blackWhiteScopeName + "." + "blacks" + "." +
discretePointScopeName;
    dropPointScope(ownerName, DiscretePointScopeItem.create(new String[]{"a",
"b"}).add("a2,*"));
}
```

高级功能介绍

我们通过上面的案例可以看到，范围有黑白名单的区分。则很容易推导出以下4个场景：

- 1. 白名单不为空，黑名单为空
- 2. 白名单不为空，黑名单不为空
- 3. 白名单为空，黑名单不为空
- 4. 白名单为空，黑名单为空

分别代表的含义，还是通过上面 招聘 的例子解释：

保安手上持有的 白名单为空，则表示任何人都可以进入公司，无论你是来应聘的还是来参观的。至于能不能接受HR 面试资格要看黑名单了。当HR 手上的 黑名单为空，则表示任何人都可以接受面试。

所以结论如下：

- 1. 白名单不为空，黑名单为空 -> 满足白名单则命中范围
- 2. 白名单不为空，黑名单不为空 -> 满足白名单，且不在黑名单内，则命中范围
- 3. 白名单为空，黑名单不为空 -> 不在黑名单内，则命中范围
- 4. 白名单为空，黑名单为空 -> 全部命中范围

创建一个黑白名单均为空的范围：

```
public static BlackWhiteScopeItem createEmptyBlackWhiteScopeItem() {
    BlackWhiteScopeItem item = new BlackWhiteScopeItem("empty_scope_test");
    /**
     * 不设置范围：白名单表示所有都通过，黑名单表示无限制 -> 公式一定成立
     */
    return item;
}
```

范例单元格	是否命中
[a2,b3,c1]	true
[a3,b3,c2]	true
[a2,b2,c1]	true
[a3,b2,c2]	true