数据平台开发规范

2020年4月14日星期二

# 概述

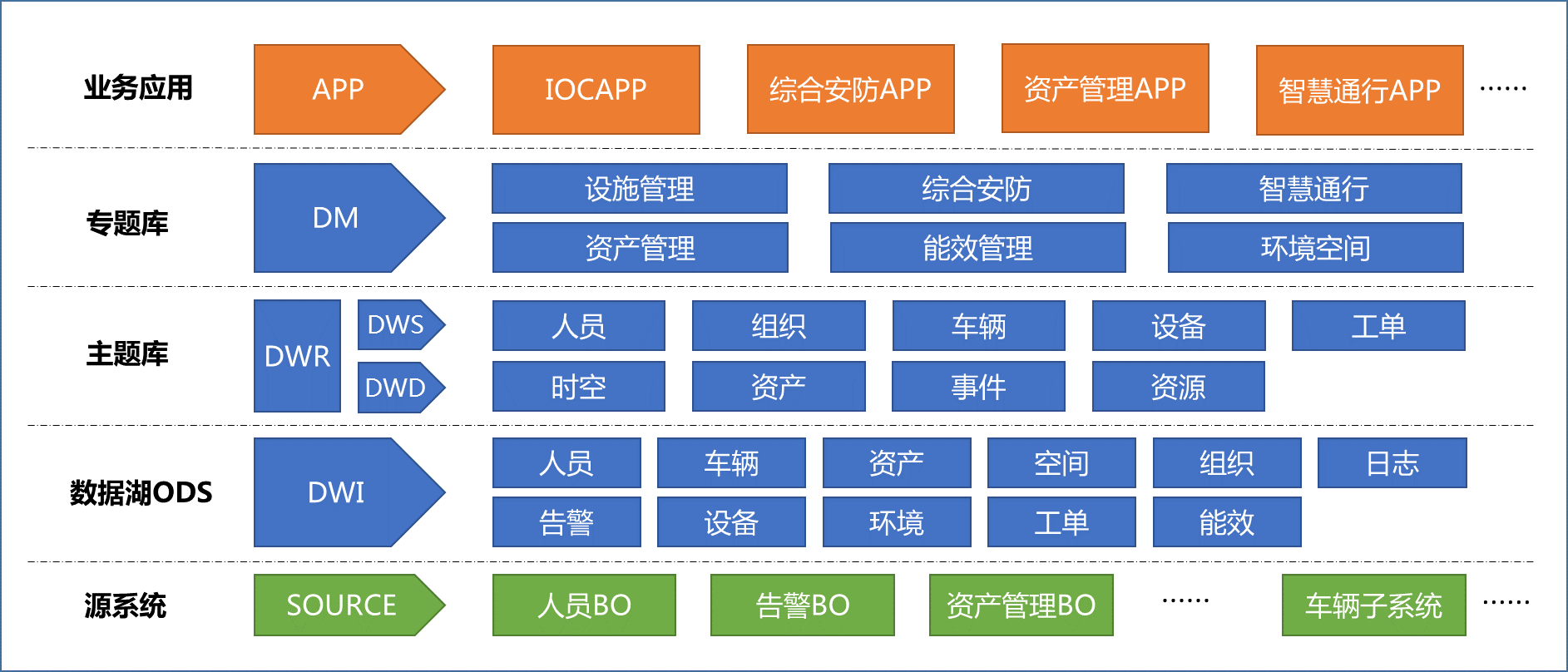
本文档描述智慧园区数据平台的各模块开发规范，包括数据模型、数据处理脚本、数据服务。

* 数据模型开发规范包括数据仓库的分层设计，各层内的领域划分、表设计规范，和模型命名规范。
* 数据处理脚本开发规范包括了批量的DLF任务开发规范、实时的CS任务开发规范。
* 数据服务开发规范包括了ROMA应用、LiveData接口的开发规范。

# 数据模型开发规范

## 数据平台模型分层架构

分析模型分为三层，专题库（DM），主题库（DWR）和数据湖（DWI）。



* APP：业务应用。
* DM（Data Market）：专题库，又叫数据集市。
* DWR（Data Warehouse Report）：主题库，又叫数据中间层。
* DWD（Data Warehouse Detail）：数据明细层。
* DWS（Data Warehouse Summary）：数据汇总层。
* DIM（Dimension）：维度模型。
* DWI（Data Warehouse Integration）：数据湖，或叫贴源层。
* SOURCE：数据源系统。

### DM Datamarket

专题库，又叫数据集市，对应专题库，即最后计算出来的报表结果，目的是使用户可以快速访问到自己关注的数据。数据集市将数据仓库中的数据按照不同业务需求进行组织和存储，面向分析和报告进行针对性设计。

#### DM层有几个特点

1. 个性化：面向不同业务需求定制开发，可复用性差。
2. 大宽表：面向查询的结果设计的报表，字段多。
3. 查询性能高：直接面向最终结果设计的报表，查询性能快。

#### DM层表设计原则

1. 面向业务应用设计模型。
2. 便于查询，尽量直接满足业务功能使用，消除关联查询，减少查询使用时间，提升性能。
3. 数据来自DWR中事实表和维度表。
4. 根据业务类型进行分类。
5. 增加审计字段。

按当前系统业务，可分为7个分类：

| **应用** | **Domain** |
| --- | --- |
| 综合安防 | security |
| 智慧通行 | access |
| 设施管理 | facility |
| 资产管理 | asset |
| 能效管理 | energy |
| 环境空间 | env |
| 智慧办公 | office |

### **DWR Data Warehouse Report**

主题库，又叫数据仓库的数据中间层。对上支撑DM层多维分析需求和明细查询需求，对下屏蔽DWI层多源的变化，并将三范式模型转为维度模型。

DWR层内又可分为DWD层和DWS层。DWD是数据明细层，DWS是数据汇总层。DWS是基于DWD的轻度汇总，例如按日、月、年进行预统计。

#### DWR层有几个特点

1. 按主题进行分类，主题依据业务进行划分。
2. 围绕业务原始行为建模，记录业务过程，不针对特定业务需求设计。
3. 维度建模，星型模型。
4. 保存最完整的、最明细的原子事实数据。
5. 记录和体现业务历史变化。
6. 模型稳定可靠，屏蔽DWI层变化。

#### DWR层表设计原则

* DWR之DWD层

1. 数据来自DWI层，但是我们的车辆、环境信息直接来自子系统。
2. 维度建模，事实表和维度表组合构成星型模型。
3. 明细层事实表需要保留最完整、最细粒度的数据。
4. 明细层表数据需要记录事实数据的历史变化过程。
5. 设计丰富详细的维度数据，满足业务多种统计需求。
6. 明细层不针对特定的业务需求设计。
7. 明细层不进行数据汇总。
8. 明细层添加审计字段。

* DWR之DWS层

1. 数据来自DWD层或者DWI层。
2. 是对数据的轻度汇总，提供通用的、常用的、稳定的基础统计数据。
3. 非定制化的、个性化的。
4. 为了查询性能，可退化维度，将维度冗余进表中，表现为宽表化。
5. 汇总层添加审计字段。

当前园区业务分为9个主题，如 [表1](https://support.huawei.com/bescloud/Smart Campus/20.0/topic/view.do?projectid=328572906&lang=zh&pidid=pid_bookmap_0175250265&topicid=TOPIC_0200882377" \l "ZH-CN_TOPIC_0200882377__table32057793)。

| **表1 主题表** | | |
| --- | --- | --- |
| **中文** | **描述** | **Domain** |
| 人员 | 人员基本信息，人员分类信息等 | person |
| 组织 | 组织结构关系，集团、部门关系等 | org |
| 设备 | 设备基本信息，点位信息，设备位置信息 | facility |
| 车辆 | 车辆基本信息，车辆与人员绑定关系等 | vehicle |
| 时空 | 建筑信息，园区位置，天气信息，城市、国家信息，特殊集团区域划分等 | space |
| 资产 | 资产信息，标签信息等 | asset |
| 事件 | 事件类型、等级等 | event |
| 资源 | 水、电表等资源信息 | res |
| 工单 | 设备维修等服务单信息 | order |

### **DWI Data Warehouse Integration**

数据湖，又叫贴源层，是按业务数据模型的统一标准和规范将源系统数据进行整合。对下屏蔽源系统不同的数据库类型和业务模型的变化，对上提供稳定的、完整的源系统数据。

#### DWI层有几个特点

1. DWI层来自园区几个业务系统提供，相对比较稳定（这点有别于业界传统模式）。
2. DWI层的数据已经经过清洗，数据质量高于源系统。
3. DWI层保留了历史数据，即使源系统可能没有保留。

#### DWI层表的设计原则

1. 按照主题类型分类。
2. 数据要求经过清洗。
3. 要求保留历史数据。

DWI分为11个库，如[表1](https://support.huawei.com/bescloud/Smart Campus/20.0/topic/view.do?projectid=328572906&lang=zh&pidid=pid_bookmap_0175250265&topicid=TOPIC_0200882317" \l "ZH-CN_TOPIC_0200882317__table38787544)所示。

| **表1 DWI库** | | |
| --- | --- | --- |
| **中文** | **描述** | **Domain** |
| 人员 | 人员基本信息，人员分类，人员进出数据等 | person |
| 组织 | 组织结构关系，集团、部门关系等 | org |
| 设备 | 设备基本信息，点位信息，设备位置，设备上报消息，告警，能耗数据信息等 | facility |
| 车辆 | 车辆基本信息，车辆与人员绑定关系，停车信息，车辆进出数据等 | vehicle |
| 空间 | 建筑信息，园区位置，天气信息，城市、国家信息，特殊集团区域划分等 | space |
| 资产 | 资产信息，标签信息，资产轨迹信息，标签上报数据等 | asset |
| 资源 | 水、电使用数据，基本资源信息 | res |
| 工单 | 设备维修等服务单信息 | order |
| 告警 | 设备、安防告警事件 | alarm |
| 日志 | 业务日志 | log |
| 环境 | 天气信息，如气温、风向、PM2.5等 | env |

## **模型命名规范**

1. 名称的长度不超过32个字符。
2. 名称采用英文单词、英文单词缩写和数字，单词之间用“\_”分隔。不得采用“\_”作为名称的起始字母和终止字母。
3. 命名能够望文知意，尽量避免使用自定义缩写（可以使用通用的术语缩写进行命名）。例如，在命名中可以使用具有实际业务含义的英文词汇。
4. 名称不得与数据库管理系统保留字冲突。
5. 不要在对象名的字符之间留空格。

### Schema规范

1. Schema命名定义为数据分层+业务主题名，例如dwi\_person。
2. Schema名全部采用小写。

### **表名规范**

1. “[Prefix]\_[domain]\_XXX\_[Suffix]”对应“前缀、领域、表义名、后缀”。
2. 表义名以其业务含义英文单词命名，多个单词间“\_”作为分割符。
3. DM层表名规范。

domain = {security、facility、env、energy、asset、access、office}

| **表类型** | **Schema** | **Prefix** | **Suffix** | **表名规则** |
| --- | --- | --- | --- | --- |
| 报告事实表 | dm\_[domain] | dm | f | [Schema].[Prefix]\_[domain]\_表义名\_[Suffix] |

1. DWR层DWD表名规范。

domain = {person、org、facility、vehicle、space、asset、event、res、order}

| **表类型** | **Schema** | **Prefix** | **Suffix** | **表名规则** |
| --- | --- | --- | --- | --- |
| 明细事实表 | dwr\_[domain] | dwd | f | [Schema].[Prefix]\_[domain]\_表义名\_[Suffix] |

1. DWR层DWS表名规范。

domain = {person、org、facility、vehicle、space、asset、event、res、order}

| **表类型** | **Schema** | **Prefix** | **Suffix** | **表名规则** |
| --- | --- | --- | --- | --- |
| 周期快照事实表（天） | dwr\_[domain] | dws | d\_f | [Schema].[Prefix]\_[domain]\_表义名\_[Suffix] |
| 周期快照事实表（周） | dwr\_[domain] | dws | w\_f | [Schema].[Prefix]\_[domain]\_表义名\_[Suffix] |
| 周期快照事实表（月） | dwr\_[domain] | dws | m\_f | [Schema].[Prefix]\_[domain]\_表义名\_[Suffix] |
| 累积快照事实表 | dwr\_[domain] | dws | h\_f | [Schema].[Prefix]\_[domain]\_表义名\_[Suffix] |

1. DWR层DIM表名规范。

domain = {person、org、facility、vehicle、space、asset、event、res、order}

| **表类型** | **Schema** | **Prefix** | **Suffix** | **表名规则** |
| --- | --- | --- | --- | --- |
| 主题维度表 | dwr\_dim | dim | d | [Schema].[Prefix]\_[domain]\_表义名\_[Suffix] |

1. DWI层表名规范。

domain = {facility、alarm、org、space、person、order、vehicle、asset、env、log、res}。

Suffix：来源是业务平台时，Suffix不需要；来源为其他子系统时，Suffix为子系统名称。

| **表类型** | **Schema** | **Prefix** | **Suffix** | **表名规则** |
| --- | --- | --- | --- | --- |
| 贴源层表 | dwi\_[domain] | dwi | 源系统命名 | [Schema].[Prefix]\_[domain]\_表义名\_[Suffix] |

### **字段命名规范**

1. 全部小写。
2. 采用有意义的列名，为实际含义的英文单词，多个单词间“\_”作为分割符。
3. 不要使用“ID”等与系统保留关键字冲突的单词作为列名。
4. 常用字段命名见[常用字段](https://support.huawei.com/bescloud/Smart Campus/20.0/topic/TOPIC_0200882375)。

### **索引命名**

1. 索引的命名为：“表名（或缩写）+\_+列名+\_idx”。
2. 其中多单词组成的属性列列名取前几个单词首字符，加末单词组成。

### **存储过程命名**

1. 存储过程命名由“sp+\_+存储过程标识(缩写)”组成。
2. 存储过程标识要以实际含义的英文单词构成，并用下划线分割各个组成部分。

### **函数命名**

函数命名由“func+\_+函数标识”组成。

# 数据处理批量脚本开发规范

## **DLF数据连接命名规范**

目前只使用两个数据连接：

* smart\_campus\_dbadmin：使用dbadmin用户，作为开发时必要的管理操作，不作为交付件。
* smart\_campus\_data\_warehouse：使用data\_service用户，所有数据仓库开发工作都使用这个数据连接。

1. 名称中仅能存在小写英文字母，并以下划线分割。
2. 以“smart\_campus\_”开头。
3. 名称有一定的自注释能力。

## **DLF脚本开发规范**

### 命名规范

#### 目录命名规范

1. 名称中仅能存在小写英文字母，并以下划线分割。
2. 名称有一定的自注释能力。

#### 目录结构规范

1. 根目录：/smart\_campus
2. 二层目录：/smart\_campus/数据仓库分层，例如：
   * /smart\_campus/dm
   * /smart\_campus/dwr
3. 三层目录：所有脚本都放在三层目录下。
   * /smart\_campus/dm/应用名称，例如：
4. /smart\_campus/dm/security
5. /smart\_campus/dm/facility
6. /smart\_campus/dm/env
7. /smart\_campus/dm/energy
8. /smart\_campus/dm/asset
9. /smart\_campus/dm/access
   * /smart\_campus/dwr/数据仓库九大主题库，例如：
10. /smart\_campus/dwr/person
11. /smart\_campus/dwr/org
12. /smart\_campus/dwr/facility
13. /smart\_campus/dwr/vehicle
14. /smart\_campus/dwr/space
15. /smart\_campus/dwr/asset
16. /smart\_campus/dwr/event
17. /smart\_campus/dwr/res
18. /smart\_campus/dwr/order
19. 管理目录：/smart\_campus\_dbadmin，目录（不创建子目录）存放管理操作的脚本，该目录下脚本不作为交付件。

#### 脚本命名规范

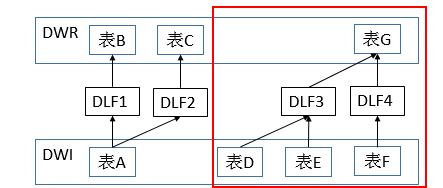
1. 只包含小写字母，并以下划线分割。
2. 以“script\_”开头，中间为目标表名（小写），以来源结尾，如：script\_{to-table-name}\_{source} 。超长的脚本名可以适当使用缩写。
3. source具有一定的自注释能力。
4. 包含的表名为加载数据的目标表名。
5. 脚本名称长度不能超过64位，名称过长的可以适当使用简写。

* to-table-name解读：

目标表的表名，如资产位置表：script\_dwr\_asset\_position\_f\_{source}。

* source解读：

在实际情况中，目标表的数据来源可能来自不同的系统，比如人员位置信息表中的数据可以来自进出门的闸机，也可以来自监控系统，那么人员位置信息表就需要从闸机表和监控表中获取，对应下图的红框中的场景：



请务必理解，该场景不同于目标表数据字段来源于不同表的字段，该场景为两张源表都可以为目标表提供一行完整的记录。

所以，我们在往人员位置信息表中加载数据时，需要两个DLF任务，一个从闸机系统读取数据，另一个从监控系统读取，我们使用source来区分，如script\_dwr\_person\_position\_f\_gate、script\_dwr\_person\_position\_f\_monitor。

#### 脚本编写规范

1. DLF功能最小、单一。

以数据来源能加载到目标表形成一条完整记录为目标，但是需要区分数据来源的类型，即上面提到的“source”。

1. 增量加载一律使用调度时间，不要使用当前时间，即${job\_plan\_time}。
2. 数据来源规范。
   * + DWI->DIM(\_D)
     + DWI、DIM、DWD->DWD(\_F)
     + DIM、DWD、DWS->DWS(\_D\_F、\_W\_F、\_M\_F、\_H\_F)
     + DIM、DWD、DWS、DM->DM(\_F)
3. 当数据来源于同层库时，只能来自同一应用的，比如dm层的数据可以来自dm层，但是安防的只能来自安防的，不能来自其他应用的，dim、dwd、dws也是这样。
4. DWS->DWS的场景：日统计数据->周、月、年统计数据等汇总事实统计。
5. DM->DM的场景：日统计数据->周、月、年统计数据等汇总事实统计。
6. 实时数据的场景待定，实时数据场景（只需要实时数据，不需要历史数据）解决方案暂时定为：dm层有一张实时表供update，DLF只执行一次，将历史数据计算后导入实时表作为初始数据，实时表的update不在DLF中，而是flink去做，所以实时场景的DLF只执行一次作为初始数据，不作为更新数据的作用。

### SQL编码排版规范

1. 字体规范：

SQL语句中关键字用大写，列名和表名采用小写。

1. 别名规范：

涉及多张表关联时每张表必须有别名，物理表别名采用“abc…”依次往后命名，中间表、临时表别名采用“t1、t2、t3…”依次往后命名

1. 格式编排：

CREATE、BEGIN、END、EXCEPTION独立成行，顶格书写不缩进，但是作为程序块嵌套在程序中时仍然需要缩进4格。

【错误示例】：

CREATE OR REPLACE FUNCTION f\_functionname (…) RETURN XXX IS

…

BEGIN

…

EXCEPTION WHEN OTHERS THEN

…

END f\_functionname;

【正确示例】：

CREATE OR REPLACE FUNCTION f\_functionname

(…

) RETURN XXX IS

…

DECLARE

BEGIN

…

EXCEPTION

WHEN OTHERS THEN

…

END f\_functionname;

1. 避免将SQL语句写到同一行，再短的语句也要在关键字和谓词处换行。

【错误示例】：

SELECT column1 INTO str\_l\_variable1 FROM t\_tablename

WHERE column2 = i\_l\_variable2

AND column3 = i\_l\_variable3;

应编写如下：

SELECT column1

INTO str\_l\_variable1

FROM t\_tablename

WHERE column2 = i\_l\_variable2

AND column3 = i\_l\_variable3;

1. SQL语句中关键字右对齐。

【错误示例】：

SELECT column1, column2, column3

INTO i\_l\_variable1, i\_l\_variable2, i\_l\_variable3

FROM t\_tablename

WHERE column4 = value

ORDER BY column2;

【正确示例】：

SELECT column1, column2, column3

INTO i\_l\_variable1, i\_l\_variable2, i\_l\_variable3

FROM t\_tablename

WHERE column4 = value

ORDER BY column2;

IMG_256

ORDER BY、GROUP BY等具有两个关键的子句，只需要第一个关键字（order、group）和其他关键字右对齐即可。

1. 缩进格数统一为4个空格，禁止使用Tab键。

缩进4格的前提是“满足SQL语句关键字的右对齐”，如例子中，SELECT相对BEGIN缩进4格，但into不能相对SELECT再缩进4格，into要与SELECT右对齐。SQL语句关键字右对齐规则的优先级较高。

1. SQL语句长度应尽量简明精炼，空格只在必要的时候才使用，不得随意添加空格，更不要出现连续多个空格的现象。关键字、保留字、连接符or、in、and等前后有一个空格，左括号之前有一个空格，右括号之后有一个空格，=、>=、<=、，(逗号)前后有有一个空格。

【错误示例】：

SELECT column1

INTO str\_l\_variable1

FROM t\_tablename

WHERE (column2 =i\_l\_variable2

AND column3>=i\_l\_variable3);

【正确示例】：

SELECT column1

INTO str\_l\_variable1

FROM t\_tablename

WHERE ( column2 = i\_l\_variable2

AND column3 >= i\_l\_variable3 );

1. IF后的条件要用括号括起来

【错误示例】：

IF i\_l\_variable1 = 2 OR i\_l\_variable2 = 4 OR i\_l\_variable3 = 6 OR i\_l\_variable4 = 8

OR i\_l\_variable5 = 10 OR i\_l\_variable6 = 12 OR i\_l\_variable7 = 14

THEN

…;

END IF;

【正确示例】：

IF ( i\_l\_variable1 = 2 OR i\_l\_variable2 = 4 OR i\_l\_variable3 = 6

OR i\_l\_variable4 = 8 OR i\_l\_variable5 = 10 OR i\_l\_variable6 = 12

OR i\_l\_variable7 = 14

) THEN

…;

END IF;

IMG_257

为让if语句更加整齐，程序的可读性强，不论含有几个条件都需要用括号括起来。另外，建议括号内的条件个数最多不要超过三个条件。

1. 对于INSERT into…VALUES，INSERT into…SELECT语句，要求一行只写一个字段。

* INSERT into…VALUES时，表名写在into关键字的后面，换行写字段名字，“（”与into关键字右对齐，“）”与“（”对齐，字段名与表名左对齐，into关键字和VALUES、SELECT关键字要右对齐。
* INSERT into…SELECT时，SELECT语句中的关键字仍然要保证右对齐，VALUES后字段必须与INSERT后字段逐行一一对应,逗号放在字段前。

【错误示例】：

INSERT INTO t\_tablename

( column1 --注释

,column2 --注释

)

VALUES

( fieldvalue1,

fieldvalue2

);

【正确示例】：

INSERT INTO t\_tablename

( column1 --注释

,column2 --注释

)

VALUES

( column1\_value --注释

,column2\_value --注释

);

【错误示例】：

INSERT INTO t\_tablename1

( column1, column2, column3, column4, column5,

column6, column7 …

)

SELECT column1\_value, column2\_value, column3\_value,

column4\_value, column5\_value, column6\_value,

column7\_value …

FROM t\_tablename2;

【正确示例】：

INSERT INTO t\_tablename1

( column1 --注释

,column2 --注释

,column3 --注释

…

)

SELECT column1\_value --注释

,column2\_value --注释

,column3\_value --注释

…

FROM t\_tablename2;

IMG_258

要求一行一个字段，保证每个字段有字段注释。

1. UPDATE语句，所有关键字要右对齐，仍然要求一行一个字段。

【错误示例】：

UPDATE t\_tablename set column1 = new\_value1,

column2 = new\_value2,

column3 = new\_value3

WHERE column4 > fix\_value1 AND column < fix\_value2;

【正确示例】：

UPDATE t\_tablename

set column1 = new\_value1, --注释

column2 = new\_value2, --注释

column3 = new\_value3 --注释

WHERE column4 > fix\_value1

AND column5 < fix\_value2;

1. 匿名块、存储过程、函数的变量定义部分，类型关键字要左对齐；变量后的注释也要左对齐。

【错误示例】：

CREATE OR REPLACE FUNCTION f\_functionname

(…

) RETURN XXX IS

str\_l\_name VARCHAR(20); --注释

i\_l\_var NUMBER; --注释

dt\_l\_time DATE; --注释

BEGIN

…

END f\_functionname;

【正确示例】：

CREATE OR REPLACE FUNCTION f\_functionname

(…

) RETURN XXX IS

str\_l\_name VARCHAR(20); -- 注释

i\_l\_var NUMBER; -- 注释

dt\_l\_time DATE; -- 注释

BEGIN

…

END f\_functionname;

1. 匿名块、存储过程、函数中的字段要左对齐；字段后的注释也要左对齐；逗号写在字段前并且逗号与字段间保留一个空格。

【正确示例】：

INSERT INTO t\_tablename1

( column1 -- 注释

, column2 -- 注释

, column3 -- 注释

…

)

SELECT column1\_value -- 注释

, column2\_value -- 注释

, column3\_value -- 注释

…

FROM t\_tablename2;

1. 条件选择语句(CASE)编写规范。

【错误示例】：

SELECT CASE WHEN column1 = column2 THEN … WHEN column3 = column4 THEN …

ELSE … END AS column5

,column6

FROM t\_tablename ;

【正确示例】：

SELECT CASE WHEN column1 = column2 THEN …

WHEN column3 = column4 THEN …

ELSE

END AS column5

,column6

FROM t\_tablename ;

1. 存储过程、函数中的IN 、OUT参数应按其类别分开书写，不允许交叉。

【错误示例】：

CREATE OR REPLACE PROCEDUREf\_functionname

(

str\_putinparam1 IN t\_tablename1.column1%type, --注释

str\_outputparam1 OUT t\_tablename1.column3%type, --注释

str\_putinparam2 IN t\_tablename1.column1%type, --注释

str\_outputparam2 OUT t\_tablename2.column2%type, --注释

str\_putinparam3 IN t\_tablename2.column3%type, --注释

str\_putinparam4 IN t\_tablename4.column1%type, --注释

str\_outputparam3 OUT t\_tablename2.column3%type, --注释

) …

【正确示例】：

CREATE OR REPLACE PROCEDUREf\_functionname

(

str\_putinparam1 IN t\_tablename1.column1%type, --注释

str\_putinparam2 IN t\_tablename1.column1%type, --注释

str\_putinparam3 IN t\_tablename2.column3%type, --注释

str\_putinparam4 IN t\_tablename4.column1%type, --注释

str\_outputparam1 OUT t\_tablename1.column3%type, --注释

str\_outputparam2 OUT t\_tablename2.column2%type, --注释

str\_outputparam3 OUT t\_tablename2.column3%type, --注释

) …

1. 注释编排：

* 匿名块、存储过程、函数要增加上注释，描述功能。
* 自定义参数需添加注释，注释与内容之间保持两个空格。
* 在逻辑相对独立的程序块之间必须用空行隔开，并加以注释。注释在内容前一行，格式对齐，单独一行。
* 脚本中关键性操作需添加注释，注释与内容之间保持两个空格。
* 脚本中所有使用到的表需添加注释，注释与内容之间保持两个空格。
* 脚本中涉及的字段需添加注释，注释与内容之间保持两个空格。

【示例】：

CREATE OR REPLACE FUNCTION f\_functionname

(…

) RETURN XXX IS

str\_l\_name VARCHAR(20); --自定义参数注释

i\_l\_var NUMBER; --自定义参数注释

dt\_l\_time DATE; --自定义参数注释

/\*

模块功能描述：

目的：

目标表 ： 表名：schema.表名; 表中文名：表中文名

源表 ： 表名：schema.表名; 表中文名：表中文名

作者 ： admin

创建日期 ： 2019-10-29

审核人 ： admin

审核时间 ： 2019-10-25

历史修改记录：

修改时间 修改作者 备注

\*/

BEGIN

…

/\*

WM\_CONCAT 自定义函数作用说明

\*/

-- 独立逻辑功能描述

Create temp table tmp\_t\_tablename1

As

SELECT a.cloumn1 --注释字段描述

,a.cloumn2 --注释字段描述

,a.cloumn3 --注释字段描述

,b.cloumn4 --注释字段描述

,b.cloumn5 --注释字段描述

to\_char(b.cloumn6,'YYYYMMDDHH24') -- 关键性操作描述

FROM t\_tablename1 a --注释表名描述

INNER JOIN t\_tablename2 b --注释表名描述

ON a.cloumn1 = b.cloumn1

;

-- 独立逻辑功能描述

INSERT INTO target\_tablename

( column1 --注释

, column2 --注释

, column3 --注释

…

)

SELECT column1\_value --注释

, column2\_value --注释

, column3\_value --注释

…

FROM tmp\_t\_tablename1 a --注释表名描述

LEFT JOIN t\_tablename3 a --注释表名描述

ON a.cloumn1 = b.cloumn1

;

END;

### SQL编码语法规范

#### 表设计规范

1. 字段设计规范

* 字段高效性原则：选择最高效的类型存储数据，这可以提高数据库的有效容量及查询执行性能。如能整型就不用浮点型，能用整型就不用字符型。
* 字段最小原则：使用满足需求的最新数值类型。如果int或smallint够用，那么选择bigint就会浪费空间。
* 当多个表存在逻辑关系时，表示同一含义的字段应该使用相同的数据类型。
* 字段最短原则：建议使用变长字符串数据类型，并指定最大长度。
* 请务必确保指定的最大长度大于需要存储的最大字符数，避免超出最大长度时出现字符截断现象。除非明确知道数据类型为固定长度字符串，否则，不建议使用CHAR(n)、BPCHAR(n)、NCHAR(n)、CHARACTER(n)。

1. **行列存设计**

* 建表时，对于有很多字段的大型表，如果单个查询只需访问较少字段，建议使用列存表。
* 如果对压缩比要求比较高，建议使用列存表。
* 数据仓库的分析型业务中，如果SELECT访问少量字段或者在少量字段上执行聚合计算，则建议使用列存表。
* 交易型业务中更新和插入频繁，建议使用行存表。
* 指定某列值的精确查询场景，建议使用行存储。
* 如果需要同时访问宽表的很多字段时，建议使用行存储。
* 如果大多数字段会出现在 SELECT 列表中或者 WHERE 子句中，建议使用行存储。
* 建表时，建议小表和维度表选择定义为复制表。
* 建表时，建议数据量较大的表定义为Hash表。

1. **分布键设计**

* 建Hash表时，指定分布建，禁止使用默认分布建。
* 数据均匀分布原则：一般情况下，选择一个能够将数据均匀分布到所有DN实例上的字段做分布键，如果单个字段不能实现数据均匀分布，则考虑使用两个字段做分布键。作为分布键的字段最好不要超过两个。使用更多的字段做分布列，会耗费更多的时间计算哈希值。
* 本地操作原则：在保证均匀分布的情况下，选择join列、group by列做分布列，可以减少计算过程中数据重分布，提升查询性能。
* 不要使用日期或者时间做分布键。

1. **分区设计**

* 为大表设置分区，小表不需要设置分区。
* 查询条件可以实现分区剪枝。
* 如果对数据有生命周期管理（如只保留一年数据，往年数据删除或者归档），需要使用分区。
* 对于列存储的表，慎用过多的分区。
* 将分区上边界的分区值定义为MAXVALUE，以防止可能出现的数据溢出。

1. **约束设计**

* 给明确不存在NULL值的字段加上NOT NULL约束，优化器会在特定场景下对其进行自动优化。
* 如果能够从业务层面补全字段值，那么，就不建议使用DEFAULT约束，避免数据加载时产生不符合预期的结果。
* 行存表支持唯一约束，而列存表不支持。
* 行存表支持主键约束，而列存表不支持。
* 行存表支持检查约束，而列存表不支持。
* 从命名上明确标识唯一约束，例如，命名为“UNI+构成字段”。

#### 语句规范

1. SELECT语句要列出具体字段名，严禁使用SELECT \* 方式。

【错误示例】：

SELECT \*

INTO str\_l\_column1, str\_l\_column2, str\_l\_column3

FROM t\_tablename

WHERE column4 = value;

【正确示例】：

SELECT column1, column2, column3

INTO str\_l\_column1, str\_l\_column2, str\_l\_column3

FROM t\_tablename

WHERE column4 = value;

1. INSERT语句要列出具体字段名，严禁使用INSERT INTO tablename VALUES ( … )方式，应使用INSERT INTO tablename ( … ) VALUES ( … )方式。

【错误示例】：

INSERT INTO t\_tablename

VALUES ( value1, value2, value3, value4);

【正确示例】：

INSERT INTO t\_tablename

( column1, --注释

column2, --注释

column3, --注释

column4 --注释

)

VALUES

( value1,

value2,

value3,

value4

);

1. 禁止使用不下推函数。

对脚本中逐段SQL进行explain，有“\_REMOTE\_TABLE\_QUERY\_”告警的即为不下推，需更换函数或写法，另外单独一段SQL下推不能保证整体SQL下推，所以还要针对所有SQL进行explain检。

1. 不同类型的操作符混合使用时，使用括号括起来进行隔离。

【错误示例】：

IF 'abc' || 'bcd' = 'abcbcd ' THEN

...

END IF;

【正确示例】：

IF ( ( 'abc' || 'bcd' ) = 'abcbcd' ) THEN

...

END IF;

1. 固定集合的，使用in（not in），动态集合使用 exists （not exists）。

【正确示例】：

SELECT column1

,column2

,column3

,column4

,column5

FROM t\_tablename;

WHERE t1.column6 NOT IN(‘1’, ‘2’, ‘3’);

SELECT first\_name

, last\_name

FROM emp a

WHERE not exists ( SELECT department\_id

FROM dept b

WHERE b.department\_id = a.department\_id

);

1. 如果没有去重的需求，优先使用UNION ALL，UNION具有去重和排序过程，union all 没有，所以速度相对比UNION要快。
2. 减少关联表的次数。
3. 排序取值和去重使用row\_number()，尽量少使用distinct、group by很多字段。
4. 脚本中不要频繁使用order by，如有必要在最后一步进行。
5. 脚本中禁止使用硬编码。
6. 不要使用太多的嵌套查询，可创建临时表。

### 实现增量加载

样例：

WITH maxtime AS (

SELECT MAX(insert\_date) FROM plan\_time\_test.plan\_time\_data\_to

)

insert into plan\_time\_test.plan\_time\_data\_to (id,name,data\_date,insert\_date) select id,name,data\_date,LOCALTIMESTAMP(0) from plan\_time\_test.plan\_time\_data\_from

where (SELECT \* FROM maxtime) < insert\_date OR (SELECT \* FROM maxtime) IS NULL;

## **DLF作业开发规范**

### 命名规范

#### 目录命名规范

1. 名称中仅能存在小写英文字母，并以下划线分割。
2. 名称有一定的自注释能力。

#### 目录结构规范

所有脚本都放在三层目录下。

| **层级** | **说明** | **示例** |
| --- | --- | --- |
| 根目录 | /smart\_campus | /smart\_campus |
| 二层目录 | /smart\_campus/数据仓库分层 | * /smart\_campus/dm * /smart\_campus/dwr |
| 三层目录 | /smart\_campus/dm/应用名称 | * /smart\_campus/dm/security * /smart\_campus/dm/facility * /smart\_campus/dm/env * /smart\_campus/dm/energy * /smart\_campus/dm/asset * /smart\_campus/dm/access |
| /smart\_campus/dwr/数据仓库九大主题库 | * /smart\_campus/dwr/person * /smart\_campus/dwr/org * /smart\_campus/dwr/facility * /smart\_campus/dwr/vehicle * /smart\_campus/dwr/space * /smart\_campus/dwr/asset * /smart\_campus/dwr/event * /smart\_campus/dwr/res * /smart\_campus/dwr/order |

#### 作业命名规范

1. 只包含小写字母，并以下划线分割。
2. 以“job\_”开头，其他部分与对应脚本名称一样。

#### 新建job填写规范



### 脚本节点配置规范

1. 数据连接一律选择“smart\_campus\_data\_warehouse”。
2. 数据库一律选择“data\_service”。
3. 节点名称与脚本名称一致。
4. 失败重试：是。
5. 最大重试次数：3。
6. 重试间隔时间（秒）：60。
7. 其他使用默认即可。

**图1 脚本节点配置图**  


### 调度配置规范

1. 目前先按照每小时的0分周期性调度。
2. 依赖属性按照来源配置。

* DWI->DIM(\_D)
* DWI、DIM、DWD->DWD(\_F)
* DIM、DWD、DWS->DWS(\_D\_F、\_W\_F、\_M\_F、\_H\_F)
* DIM、DWD、DWS、DM->DM(\_F)

如：DWS中的表依赖了DIM、DWD中的表，所以当配置DWS中表的调度时，就需要配置两个依赖。

1. 依赖的作业失败后，当前作业处理策略选择“挂起”。
2. “等待依赖作业的上一周期结束，才能运行”不勾选。
3. “跨周期依赖”选择“自依赖（等待上一调度周期结束才能继续运行）”。

### 调试

1. 调试时调度周期可以设置得短一点。
2. 调试之前需要有新数据在依赖的底层的表中加载，否则调试视为无效。
3. 开发完成后按照规范调整调度策略，新数据加载的调度可适当提前，需保证有新数据加载。

## **数据批量采集开发规范**

### BingoETL开发规范

当前通过BingoETL做数据从业务平台到数据平台的批量采集。BingoETL的任务需要优先于DLF的任务执行。举例来说BingoETL任务在每小时的05分执行，采集需要花费5分钟，那么后续DLF任务可以在每小时的15分启动，保障BingoETL已经采集完成数据。

需要注意的是，当前版本BingoETL不支持设置定时任务的起始时间，目前是以点击启动任务的时间作为每个周期的起始时间的。所以这个版本开发时，需要手动控制任务启动时间，来保证和DLF之间的执行顺序正确。

#### 目录命名规范

1. 目录结构名称中仅能存在小写英文字母，并以下划线分割。
2. 名称有一定的自注释能力。

#### 规范

| **层级** | **说明** | **示例** |
| --- | --- | --- |
| 根目录 | /smart\_campus | /smart\_campus |
| 二层目录 | /smart\_campus/数据仓库分层 | /smart\_campus/dwi |
| 三层目录 | /smart\_campus/dwi/数据仓库九大主题库 | * /smart\_campus/dwi/person * /smart\_campus/dwi/org * /smart\_campus/dwi/facility * /smart\_campus/dwi/vehicle * /smart\_campus/dwi/space * /smart\_campus/dwi/asset * /smart\_campus/ dwi/alarm * /smart\_campus/ dwi /res * /smart\_campus/ dwi /order |

#### 转换作业命名规范

1. 只包含小写字母，并以下划线分割。
2. 以“trans\_”开头，以目标表结尾，如：trans \_{to-table-name}。超长的脚本名可以适当使用缩写。
3. to-table-name具有一定的自注释能力。

#### 作业开发规范

1. 作业命名规范
2. 只包含小写字母，并以下划线分割。
3. 以“job\_”开头，以目标表结尾，如：job \_{to-table-name}。超长的脚本名可以适当使用缩写。
4. to-table-name具有一定的自注释能力。
5. 作业开发规范

作业中转换组件的命名需跟使用的转换作业名字一致。

### FDI开发规范

#### ROMA应用的命名规范

dataservice.etl.xxx（其中xxx为贴源层的名称，人员：person，车辆：vehicle，告警：alarm，设备：facility，工单：order，资产：asset，资源：res，时空：space，环境：env，组织：org）。

#### FDI的任务名称命名规范

* 实时任务：
  1. 只包含小写字母，并以下划线分割。
  2. 以“job\_real”开头，以cs作业任务名为依据，如：
     + cs作业为：cs\_res\_datalog\_all
     + FDI实时任务为：job\_real\_res\_datalog\_all
* 批量任务：
  1. 只包含小写字母，并以下划线分割。
  2. 以“job\_batch”开头，以bingoETL作业任务名为依据，如：
     + bingoETL作业为：job\_dwi\_person\_type
     + FDI实时任务为：job\_batch\_dwi\_person\_type

#### FDI的MQS节点名称命名规范

统一写当前任务对应的TOPIC名称。

#### FDI的API节点名称命名规范

统一写：read

#### FDI的transform节点名称命名规范

1. 只包含小写字母，并以下划线分割。
2. 以“transform\_”开头，以目标表结尾，如：作业往目标表dwi\_person\_type写入数据，transform节点命名为：transform\_dwi\_person\_type。

#### FDI的GaussDB 200节点名称命名规范

1. 只包含小写字母，并以下划线分割。
2. 以“write\_”开头，以目标表结尾，如：作业往目标表dwi\_person\_type写入数据，GaussDB 200节点命名为：write\_dwi\_person\_type。

## **数据处理实时消息任务开发规范**

### MQS开发规范

DataService的交付件只涉及北向MQS的配置，不包含MQS消息的接入。

#### 命名规范

MQS名称以“T\_XXX\_MSGOUT”为原型输出，大写，以“\_MSGOUT”结尾，XXX为具体场景。

### CS开发规范

#### 命名规范

CS作业命名规范：job\_cs\_{模块名}\_作用，描述里要写清楚作业的具体作用，并且脚本里要有关键注释。

#### 存放路径

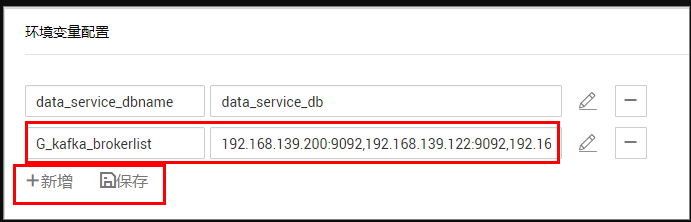
/smart\_campus/data\_service/{模块名称}

#### 全局变量配置及使用

1. 全局变量命名。

命名规范：G\_xxx，例如：G\_kafka\_brokerlist。

所有公共变量、app秘钥等都要配置为全局变量。



1. 全局变量引用。

引用方式：#{Env.get("G\_kafka\_brokerlist")}

CREATE SOURCE STREAM sensor\_monitoring\_infos (eventTime TIMESTAMP, deviceId STRING, temperature DOUBLE, humidity DOUBLE, pm25 DOUBLE, noise DOUBLE)

WITH (

type = "kafka",

kafka\_bootstrap\_servers = '#{Env.get("G\_kafka\_brokerlist")}',

kafka\_group\_id = "whgroup1",

kafka\_topic = "T\_ENV\_Source",

encode = "json",

json\_config = "eventTime=eventTime;deviceId=deviceId;temperature=data.InfoService\_Temperature;humidity=data.InfoService\_Humidity;pm25=data.InfoService\_PM25;noise=data.InfoService\_Noise"

) TIMESTAMP BY proctime.proctime;

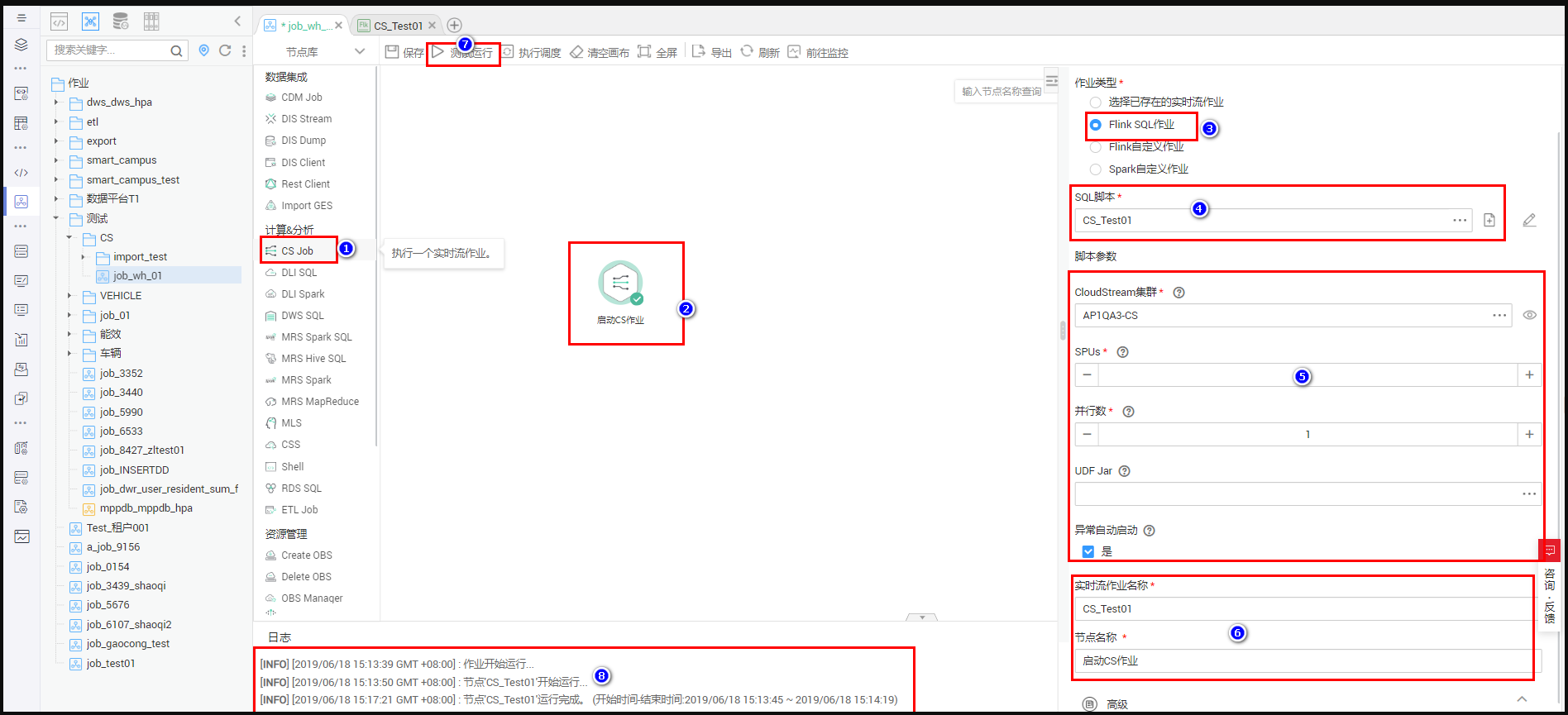
1. 注意事项。

变量表达式使用单引号括起来，表达式中变量名使用双引号括起来。

#### 任务配置&调度配置

1. DLF任务创建。

作业名称和节点名称保持一致。可以与CS作业名称一致。UDF jar路径配置为：#{Env.get("G\_obs\_flink\_jar\_path")}。

**图1 作业配置**  


1. 调度配置。

**图2 调度配置**  

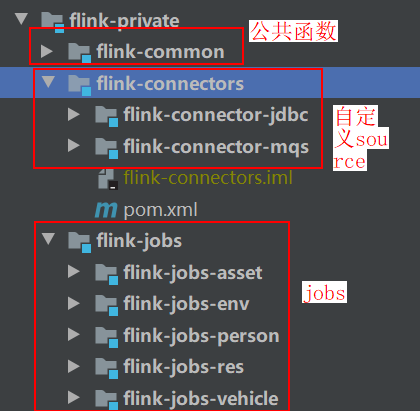

#### 归档

* 脚本归档位置：**ParkCore\DataService\dlf\scripts**
* 作业归档位置：**ParkCore\DataService\dlf\jobs**

### **Flink(java)开发规范**

#### 实时代码开发规范

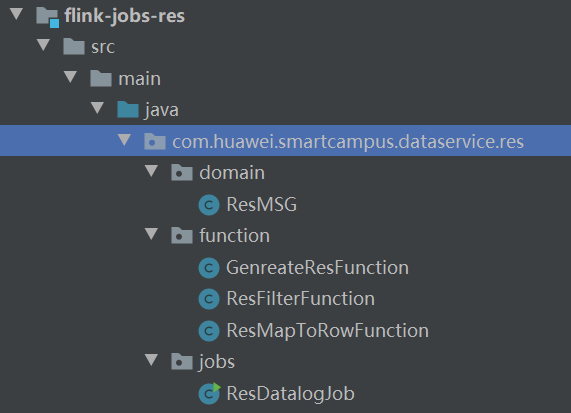
* 模块命名规范
  1. 每个模块是一个maven子工程，父工程为flink-private。
  2. 公共函数工具类等放在flink-private/flink-common子工程下。
  3. 自定义source等连接器放在flink-private/flink-connectors下，目录命名参考flink-connector-jdbc。
  4. Job工程放在flink-private/flink-jobs/flink-jobs-{模块名}下。



* 包命名规范

任务包名可根据实际情况自行定义。最底层的三个目录建议分层：

* 1. domain:存放任务用到的消息对应的Java POJO对象。
  2. function：存放flink任务处理过程中的自定义转换函数。
  3. jobs：存放本模块下所有的flink任务。



* 类命名规范
  1. 任务的job入口类，命名以**{业务语义}Job**为佳，例如：ResDatalogJob，PeopleHeatJob
  2. 任务用到的自定义Function，建议名称为**{功能}Function**，例如：ResFilterFunction
  3. Java POJO类命名可根据消息类型自行定义。
* 其他规范
  1. 代码中需要方法要给出注释。
  2. 关键步骤要添加注释。
  3. 关键处需要打印日志，利于定位问题。
  4. 异常要打印日志，及时处理。
  5. 变量命名要规范，不要随意命名。

#### Java代码通用规范

##### 代码风格

1. 命名

* 为包、类、方法、变量取一个合适的名字，使代码易于理解。
* 禁止使用魔鬼数字
* 直接使用数字，造成代码难以理解，也难以维护。应采用有意义的静态变量或枚举来代替。
* 例外情况，有些特殊情况下，如循环或比较时采用数字0，-1，1，这些情况可采用数字。
* 常量命名，由全大写单词组成，单词间用下划线分隔，且使用 static final修饰。
* 变量、属性命名，使用名词，并采用首字母小写的驼峰命名法。
* 方法的命名，用动词和动宾结构，并采用首字母小写的驼峰命名法，格式如下：
  + - * get + 非布尔属性名()
      * is + 布尔属性名()
      * set + 属性名()
      * has + 名词/形容词()
      * 动词()
      * 动词 + 宾语()
* 类和接口的命名，采用首字母大写的驼峰命名法。
* 包的命名，由一个或若干个单词组成，所有的字母均为小写。

1. 注释

* 注释应解释代码的意图，而不是描述代码怎么做的。
* 保证注释与代码一致，避免产生误导。
* 注释应与其描述代码位置相邻，放在所注释代码上方或右方，并与代码采用同样缩进。
* 一般单行注释用//，块注释用/\* \*/，JavaDoc注释用/\*\* \*/

1. 排版

* 团队应遵守一致的排版风格。
* 将排版风格固化到IDE的代码格式化配置文件中，并让整个团队使用。
* 在不同的概念之间，增加空行。
* 将逻辑紧密相关的代码放在一起。
* 控制一行的宽度，不要超过120个字符。
* 在不同的概念间（关键字、变量、操作符等）增加空格，以便清楚区分概念。
* 采用缩进来区分不同层次的概念。
* 将局部变量的作用域最小化。
* 给if、for、do、while、switch等语句的执行体加大括号{}。
* 控制文件的长度，最好不要超过500行。

##### 变量和类型

1. 避免随意进行类型强制转换，应改善设计，或在转换前用instanceof进行判断。
2. 需要精确计算时不要使用float和double。
3. 不能用浮点数作为循环变量。
4. 浮点型数据判断相等不能直接使用==。
5. 避免同一个局部变量在前后表达不同的含义。
6. 不要在单个的表达式中对相同的变量赋值超过一次。
7. 基本类型优于包装类型，注意合理使用包装类型。

##### 方法

1. 方法设计的第一原则是要短小。
2. 方法设计应遵循单一职责原则（SRP），一个方法仅完成一个功能。
3. 方法设计应遵循单一抽象层次原则（SLAP）。

IMG_256

SLAP原则，是指让一个方法中所有的操作处于相同的抽象层。否则跳跃的代码的抽象层次破坏了代码的流畅性。

1. 方法设计应遵循命令与查询职责分离原则（CQRS）。
2. 不要把方法的入参当做工作变量/临时变量，除非特别需要。
3. 应明确规定对接口方法参数的合法性检查由调用者负责还是由接口方法本身负责。
4. 方法的参数个数不宜过多，不要超过七个。
5. 谨慎使用可变数量参数的方法。

##### 异常和日志

1. 异常

* 只针对真正异常的情况才使用exception机制；不要用exception机制来做流程控制。exception机制只应该用于处理罕见的、意料之外的、导致正常流程无法继续执行的行为，而不是取代正常的业务逻辑判断。
* 在抛出异常的细节信息中，应包含能捕获失败的信息。
* 对可恢复的情况使用受检异常（checked exception），对编程错误使用运行时异常（runtime exception）。
* 方法注释和文档中要包含所抛出异常的说明。
* 对第三方API抛出大量各类异常进行封装。
* 在finally块中不要使用return、break或continue使finally块非正常结束。
* 不要直接捕获受检异常的基类Exception。
* 一个方法不应抛出太多类型的异常。

1. 日志

* 日志信息准确、繁简得当，满足快速定位的需要。
* 日志的记录，不要使用 System.out 与 System.err 进行控制台打印，应该使用专用的日志工具(比如：slf4j+logback)进行处理。
* 日志工具对象logger应声明为private static final。
* 日志应分等级；日志可分为以下级别：debug、info、warn、error、fatal。
* 日志中不要记录敏感信息。

## **数据服务开发规范**

### ROMA应用开发规范

#### ROMA应用的命名规范

1. dataservice.api.xxx

其中xxx为专题库的名称 (security、facility、env、energy、asset、access、office)。

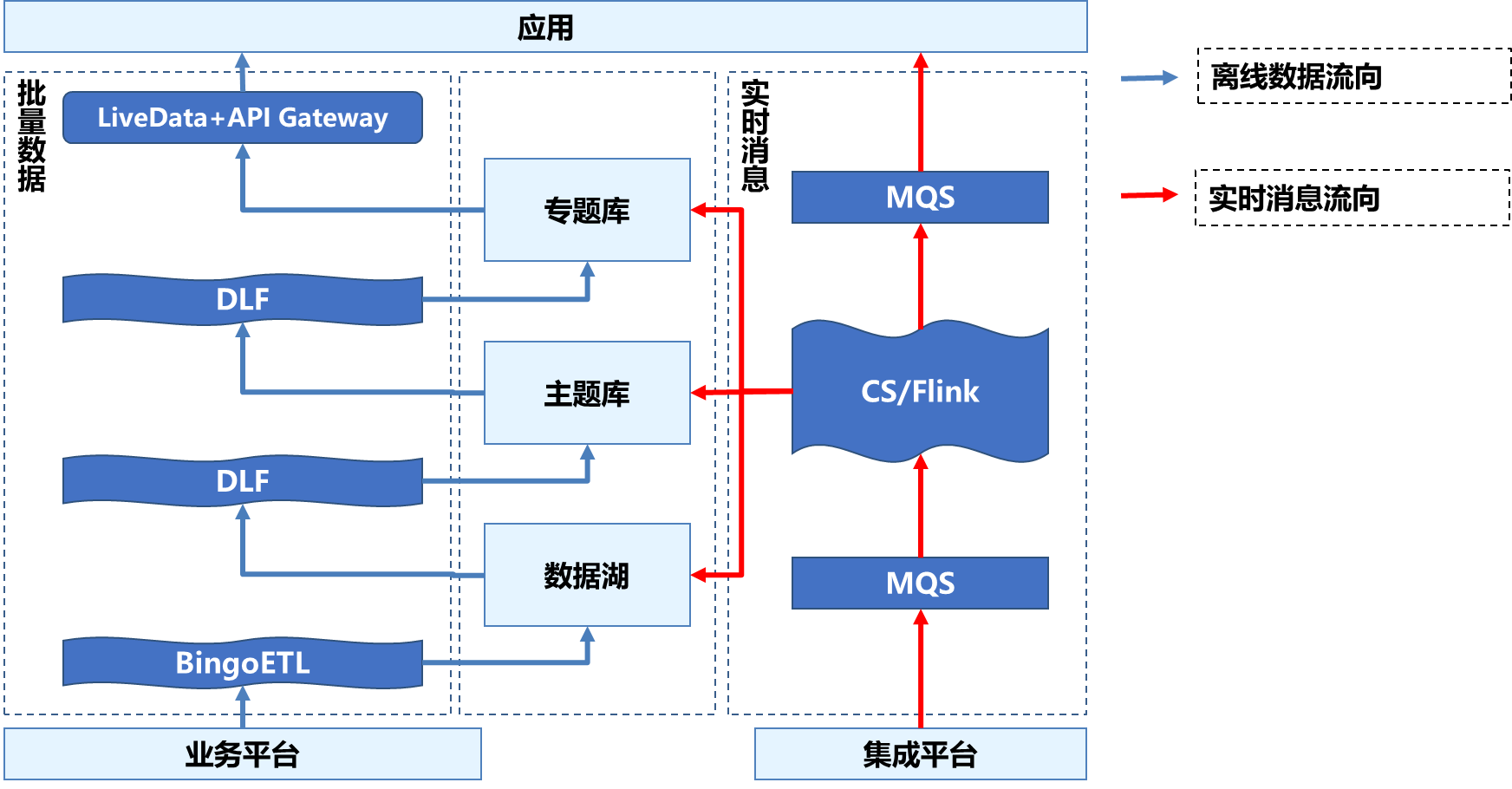
1. dataservice.msg.yyy

其中yyy为11个贴源层的名称（facility、alarm、org、space、person、order、vehicle、asset、env、log、res），如果不需要实时消息，则不创建。

#### ROMA中不同应用的开发内容

1. dataservice.api.xxx仅仅包含LiveData，为上层应用提供服务。
2. dataservice.msg.yyy包含MQS配置。

数据流向图如[图1](https://support.huawei.com/bescloud/Smart Campus/20.0/topic/view.do?projectid=328572906&lang=zh&pidid=pid_bookmap_0175250265&topicid=TOPIC_0200882365" \l "ZH-CN_TOPIC_0200882365__fig2196111572211)，数据平台负责虚线框中的部分：

**图1 数据流向图**  


### LiveData开发规范

#### URL命名规范

1. 统一采用“/dataservice/api/模块名/关键词”的形式。
2. 模块名与关键词规范。
   * 名词组合。
   * 各种名词以“-”分割。
   * 单词一律小写。
   * 模块名为应用名称（security、facility、env、energy、asset、access、office）。
   * 关键词有一定的自注释能力。

#### LiveData数据来源优先级

DM>DWS>DWD>DWI

#### LiveData命名规范

1. 名称有一定的自注释能力。
2. 名称使用动宾结构。
3. 统计类接口以“（时间）”结尾，如统计在园人数（实时）、统计车辆通过数量（天）、统计用电量（月）。
4. 描述信息是对接口的解释，让读者知道接口的作用。

#### HTTP方法规范

使用GET方法。

#### 请求参数规范

1.具有一定的自注释能力。

2.单词一律小写，单词之间以“\_”连接。

3.例如“园区编码：park\_code”。

#### 返回类型规范

使用JSON。

#### LiveData脚本

1. SQL嵌套不能超过2层。
2. 不能使用“select \*”，使用具体字段。
3. 关键逻辑添加注释。
4. 尽量减少函数的使用。

## **附录**

# 常用字段

| **字段** | **含义** | **类型** | **长度** |
| --- | --- | --- | --- |
| device\_id | 设备id | character varying | 100 |
| device\_code | 设备code | character varying | 255 |
| park\_code | 园区code | character varying | 100 |
| park\_name\_cn | 园区中文名 | character varying | 500 |
| count\_entry | 进园数 | integer | - |
| count\_exits | 出园数 | integer | - |
| report\_date | 记录日期（天） | timestamp without time zone | - |
| record\_time | 工单记录时间 | timestamp without time zone | - |
| alarm\_level\_info | 提示 | character varying | 100 |
| alarm\_level\_normal | 一般 | character varying | 100 |
| alarm\_level\_major | 重要 | character varying | 100 |
| alarm\_level\_critical | 严重 | character varying | 100 |
| entry\_time | 进场时间 | timestamp without time zone | - |
| exits\_time | 出场时间 | timestamp without time zone | - |
| gender | 性别 | character varying | 50 |
| person\_name | 人员姓名 | character varying | 100 |
| asset\_code | 资产编码 | character varying | 255 |
| nationality | 国籍 | character varying | 255 |
| effective\_date | 生效时间 | timestamp without time zone | - |
| failure\_time | 失效时间 | timestamp without time zone | - |
| expiration\_date | 截止时间 | timestamp without time zone | - |
| complete\_time | 完成时间 | timestamp without time zone | - |
| definition\_code | 规格编码 | character varying | - |
| category\_name/type\_name | 分类名称 | character varying | 100 |
| action\_type | 操作类型 | character varying | 100 |
| brand | 产品品牌 | character varying | 255 |
| manufacturer | 产品厂商 | character varying | 255 |
| status | 产品状态 | character varying | 100 |
| description | 描述 | character varying | 500 |
| type\_description\_lang | 全类型描述 | character varying | 500 |
| path | 路径 | character varying | 500 |
| date\_of\_birth | 出生日期 | timestamp without time zone | - |
| direction | 人员进出：   * In：进 * Out：出 | character varying | 50 |
| passing\_result | 通行结果 | character varying | 100 |
| space\_code | 空间代码 | character varying | 100 |
| definition\_code | 规格编码 | character varying | 255 |
| altitude | 海拔 | character varying | 100 |
| longitude | 经度 | character varying | 100 |
| latitude | 纬度 | character varying | 100 |
| line\_code | 进线编码 | - | 100 |
| dw\_creation\_by | 数据创建者 | character varying | 100 |
| dw\_creation\_date | 数据创建时间 | timestamp without time zone | - |
| dw\_last\_update\_by | 数据最后更新者 | character varying | 100 |
| dw\_last\_update\_date | 数据最后更新时间 | timestamp without time zone | - |
| dw\_batch\_number | 数据批次号 | Bigint | - |