Alibaba Cloud DataWorks

データの開発

Document Version20191205

目次

1 データの開発

1.1 ソリューション

データ開発モードは、3 階層構造 (プロジェクト - ソリューション - ビジネスフロー) へアップグレードされました。従来のディレクトリ編成モードは使用しません。

プロジェクト - ソリューション - ビジネスフロー (Project - solution - business flow)

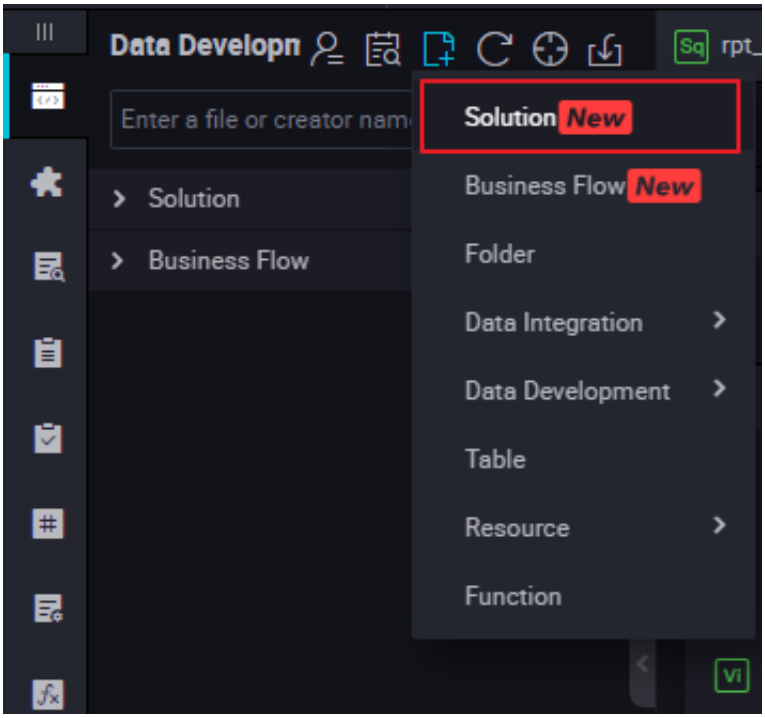
DataWorks の新バージョンでは、ビジネスタイプに基づいて、異なるタイプのノードタスクを統合できるように、データ開発モードがアップグレードされました。 このような構造を使用することで、ビジネスごとのコード開発がさらに容易になります。 開発プロセスでは、より広い視野で複数のビジネスフローにまたがる開発が実現します。 ユーザーの開発エクスペリエンスを向上させるために、開発プロセスはプロジェクト - ソリューション - ビジネスフロー **(Project - solution - business flow)** の 3 階層構造に基づいて再定義されます。

- プロジェクト**:** 権限組織の基本単位で、ユーザーの権限 (開発や **O&M** 権限) を制御するために使用します。 同じプロジェクト内では、プロジェクトメンバーのすべてのコードを共同で開発、管理することができます。
- ソリューション**:** ビジネスフローを組み合わせて、ソリューションをカスタマイズします。 利点**:**
  - 1 つのソリューションが複数のビジネスフローから構成されます。
  - 同じビジネスフローを別のソリューションで再利用することができます。
  - 没入型開発を組み合わせたソリューションへ実装することができます。
- ビジネスフロー**:** ビジネスの抽象エンティティです。ビジネスの視点でデータコード開発を整理することができます。 ビジネスフローは、複数のソリューションで再利用できます。 利点**:**
  - ビジネスフローでは、ビジネスの視点からコードを整理することができます。 タスクタイプベースのコード編成モードがあります。 複数階層のサブディレクトリをサポートします(4 階層までを推奨) 。
  - ビジネスの視点からワークフロー全体を確認し、最適化することができます。
  - ビジネスフローダッシュボードを使用することで、開発効率が向上します。
  - ビジネスフローに基づいて、リリースや **O&M** を編成することができます。

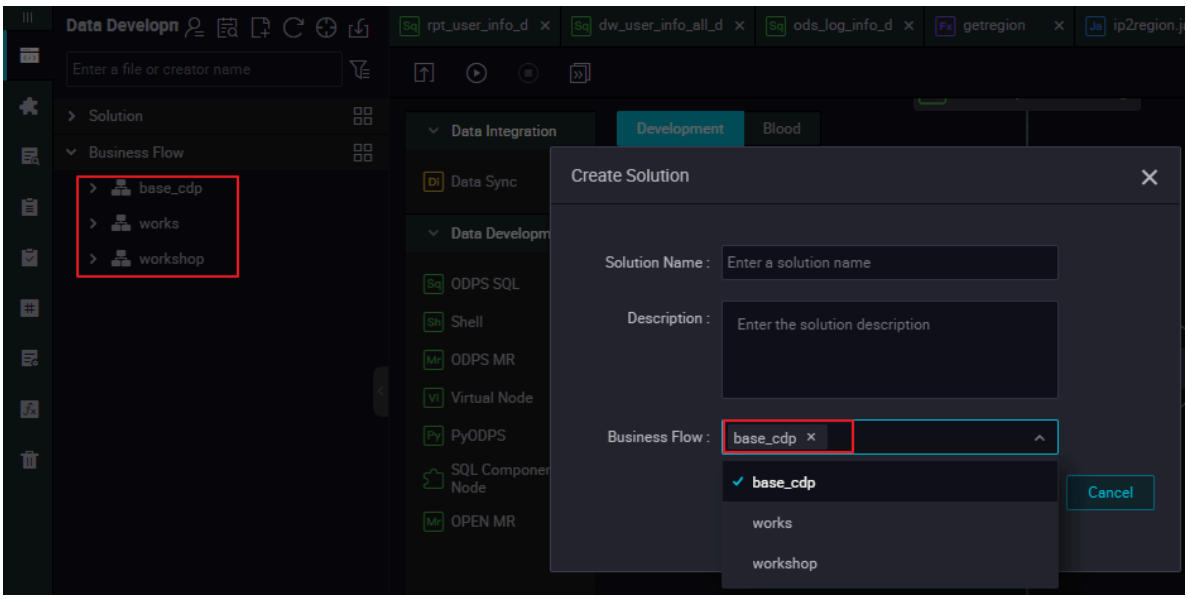
没入型開発エクスペリエンス

作成したソリューションをダブルクリックすると、開発エリアからソリューションエリアへ切り替わります。 現在のソリューションの内容のみが、ディレクトリに表示されます。 最新の環境が提供されるため、現在のソリューションと関連のないプロジェクトのコードに煩わされることがありません。

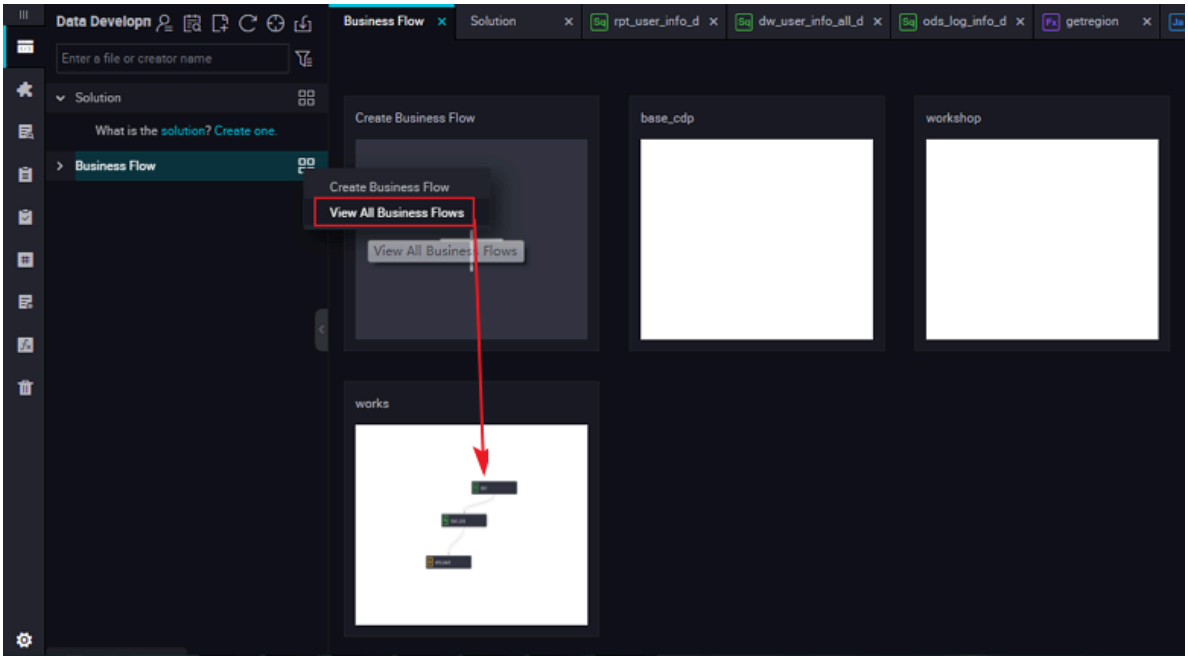
1. **[DataStudio]** ページへ移動し、ソリューションを作成します。



2. 作成したソリューションから閲覧するビジネスフローを選択します。



3. 選択したビジネスフローのノードを閲覧する場合、またはソリューションを変更する場合は、**[View All Business Flows]** を右クリックします。



4. 別のページへ移動します。

   - **[Publish]** をクリックすると、**[Task Publish]**ページへ移動します。 現在のソリューションで **[To be released]** となっているノードがこのページに表示されます。
   - **[O&M]** をクリックすると、**[O&M Center]** > **[Periodic Instances]** ページへ移動します。 現在のソリューションにあるすべてのノードの周期インスタンスがデフォルトでこのページに表示されます。

   ビジネスフローは、複数のソリューションで再利用できます。 独自のソリューションの開発に集中することができます。 他のソリューションやビジネスフロー内で参照したソリューションの編集を別のユーザーが直接行うことができ、共同開発が実現します。

1.2 SQL コードに関するエンコードの原則と基準

エンコードの原則

SQL コードは次のとおりエンコードします。

- コーディングの原則
- コード行は明確で、きちんとした見栄えのものにします。
- コード行はきれいに配置し、正しい階層構造にします。
- コードの可読性を向上させるために、必要に応じてコメントを入力します。

- この規則の要件は開発者のコーディング作業を制約するものではありません。 実際には、一般的な要件に違反がないという前提で、コード開発にとってメリットがある場合は、この規則から合理的に逸脱しても構いません。この規則は、継続して改善および補足される予定です。
- **SQL** コードに使用するキーワードや予約済みワードには小文字を使用します。 これらのワードの例としては、**select**、**from**、**where**、**and**、**or**、**union**、**insert**、**delete**、**group**、**having** および **count** があります。
- **SQL** コードに使用するキーワードと予約済みコードに加えて、その他のコード (フィールド名やテーブル別名など) も小文字にします。
- スペース **4** つ分は、**1** インデント単位と同等です。 すべてのインデントはインデント単位の倍数の整数とし、コードの階層に応じて整列させます。
- **"select * operation"** の使用は禁止されています。 列名はすべての操作で指定する必要があります。
- 対応するラケットは同じ列に記述する必要があります。

SQL コーディングの仕様

SQL コードの仕様は次のとおりです。

- コードヘッダー

  情報カテゴリ、関数の説明、作成者、日付などの情報をコードヘッダーへ追加します。 ログとタイトルバーは、後にユーザーがレコードを変更する際に追加できるよう確保しておきます。 各行は、**80** 文字を超えることができません。 テンプレートは次のとおりです。


```
-- MaxCompute(ODPS) SQL
--
********************************************************************************
-- Subject: Transaction
-- Function description: Transaction refund analysis
 -- Author: With code
 -- Create time: 20170616
 -- Change log:
 -- Modified on  Modified by  Content
-- yyyymmdd name comment
-- 20170831 Without code Add a judgment on the transaction biz_type=
1234
```

```
--
********************************************************************************
```

- フィールドの配置要件
  - **"select"** 文で選択するフィールドに対して、フィールドごとに **1** 行使用します。
  - **"select"** の次にインデントを **1** つ入れてから、最初に選択したフィールドを続けます。 したがって、行の先頭から インデント **2** つ分先からフィールドを記述します。
  - その他の各フィールドは、インデント **2** つ目から始め、コンマ **(,)** の後にフィールド名を記述します。
  - **2** つのフィールド名の間にあるコンマ **(,)** は、**2** つ目のフィールドの直前に記述します。
  - **"as"** 文は、関連するフィールドと同じ行に記述します。 複数のフィールドに **"as"** 文を使用する場合、同じ列に配置することを推奨します。

```
select  channel_id            as channel_id
        ,trade_channel_desc   as trade_channel_desc
        ,trade_channel_edesc  as trade_channel_edesc
        ,inst_date            as inst_date
        ,trade_iswap          as trade_iswap
        ,channel_type         as channel_type
        ,channel_second_desc  as channel_second_desc
from    (
```

- **"Insert"** サブ文の配置要件

  "Insert" サブ文は同じ行に記述する必要があります。 改行は禁止されています。

- サブ文の配置要件を選択します。

  "select" 文に使用されるサブ文 **(from**、**where**、**group by**、**having**、**order by**、**join**、および **union)** は、次の要件に準拠する必要があります。

  - 改行します。
  - サブ文は、**"select"** 文に対して左揃えにします。
  - サブ文とその後に続くコードの最初の文字の間にインデントを **2** つ入れます。
  - **"where"** サブ文の論理演算子 **("and"** や **"or")** は **where** に対して左揃えにします。
  - サブ文の長さがインデント **2** つ分を超える場合、サブ文の後にスペースを追加し、後続のコード **("order by"** や **"group by")** を記述します。

```
select    trim(channel) channel
          ,min(id)      id
from      ods_trd_trade_base_dd
where     channel is not null
and       dt = ${tmp_uuuummdd}
and       trim(channel) <> ''
group by  trim(channel)
order by  trim(channel)
```

・ 演算子の前後につけるスペースの要件は、算術演算子と論理演算子の前後に 1 つずつ入れる必要があります。 1 行の長さが 80 文字を超えない限り、演算子は同じ行に記述します。

```
select      trim(channel) channel
            ,min(id)      id
from        ods_trd_trade_base_dd
where       channel is not null
and         dt = ${tmp_uuuummdd}
and         trim(channel) <> ''
group by    trim(channel)
order by    trim(channel)
```

・ **"case"** 文の記述

"select" 文では **"case"** 文を使用してフィールド値を判断したり割り当てます。 コード行の可読性を向上させるためには、**"case"** 文を正しく記述することが重要です。

以下は、**"case"** 文の記述に関する規則です。

- **"when"** サブ文は、 **"case"** 文と同じ行に記述し、インデントを 1 つ入れます。
- **"when"** サブ文ごとに 1 行使用します。 文が長すぎる場合は、改行を行うことができます。
- **"case"** 文には、**"else"** サブ文が含める必要があります。 **"else"** サブ文は、**"when"** サブ文に対して左揃えにします。

```
,case   when p1.trade_from = '3008' and p1.trade_email is null then 2
        when p1.trade_from = '4000' and p1.trade_email is null then 1
        when p9.trade_from_id is not null then p9.trade_from_id
 end                            as trade_from_id
,p1.trade_email                 as partner_id
```

・ 入れ子クエリの記述に関する仕様

入れ子サブクエリは、データウェアハウスシステムの **ETL** 環境でよく使用されます。 したがって、階層状にコードを配置することが重要です。 例**:**

```
select     p.channel
           ,rownumber()  order_id
from       (
               select   s1.channel
                        ,s1.id
               from     (
                            select  trim(channel)      as channel
                                    ,min(id)           as id
                            from    ods_trd_trade_base_dd
                            where   channel is not null
                            and     dt = ${tmp_yyyymmdd}
                            and     trim(channel) <> ''
                            group by trim(channel)
                        ) s1
               left outer join
                       dim_trade_channel s2
               on      s1.channel = s2.trade_channel_edesc
               where   s2.trade_channel_edesc is null
               order by id
           ) p
;
```

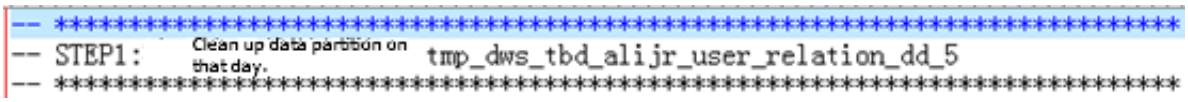
・ テーブル別名の定義規則

- 別名をすべてのテーブルに追加する必要があります。 **"select"** 文で操作テーブルに対し別名を定義した後、そのテーブルを参照する文を記述する際、常にその別名を使用する必要があります。 コードの記述を容易に行うには、別名は可能なかぎりシンプルで簡潔なものにし、キーワードの使用は避けます。
- テーブルの別名はシンプルな文字を使って定義します。 別名はアルファベット順に定義することを推奨します。
- 別名の多層入れ子サブクエリの前に、階層を示す必要があります。 **SQL** 文の別名はレイヤー別に定義します。 レイヤー **1** から **4** まではそれぞれ **P (Part)**、**S (Segment)**、**U (Unit)**、**D (Detail)** と表現します。 あるいは、レイヤー **1** から **4** までを、**a**、**b**、**c**、**d** と表現することもできます。同じレイヤーにあるサブ文を区別するために、レイヤーを示す

文字の後ろに番号 (1、2、3、4 など) を付けます。 必要に応じて、テーブルの別名にコメントを追加することができます。

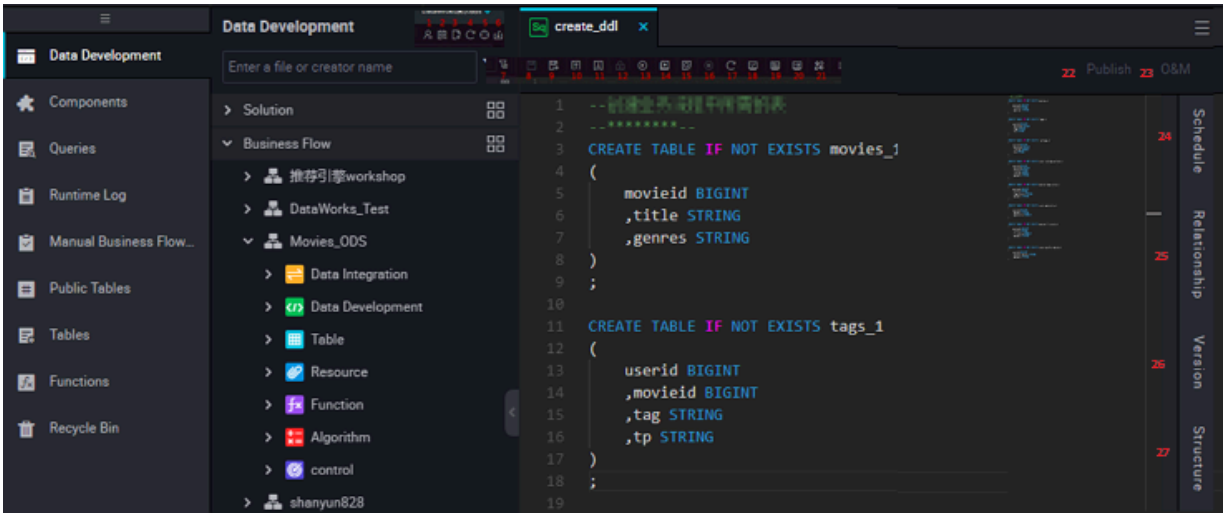
```
select     p.channel
           ,rownumber()  order_id
from       (
                select   s1.channel
                         ,s1.id
                from     (
                                  select  trim(channel)      as channel
                                          ,min(id)           as id
                                  from    ods_trd_trade_base_dd
                                  where   channel is not null
                                  and     dt = ${tmp_yyyymmdd}
                                  and     trim(channel) <> ''
                                  group by trim(channel)
                          ) s1
                left outer join
                        dim_trade_channel s2
                on      s1.channel = s2.trade_channel_edesc
                where   s2.trade_channel_edesc is null
                order by id
           ) p
;
```

- SQL コメント
  - 各 SQL 文に対してコメントを追加する必要があります。
  - 各 SQL 文のコメント用に 1 行使用し、文の前に配置します。
  - フィールドのコメントは、フィールドの次に記述します。
  - 簡単に理解できない分岐条件の場合は、コメントを追加します。
  - 関数を記述するための重要な計算に対してもコメントを追加します。
  - 関数が長すぎる場合は、実装する関数に基づいて文をセグメント化し、それぞれのセグメントを説明するコメントを追加します。
  - 定数または変数の場合は、保存する値の意味を説明するコメントを追加します。値の有効範囲を説明するコメントも任意に追加できます。



1.3 コンソールの機能

1.3.1 コンソールの紹介



インターフェイス機能についての説明は次のとおりです。

No.	機能	説明
1	**Show My Files**	クリックして、現在の列で、ご自身のアカウントにあるノードを表示します。
2	**Code Search**	クリックして、コードまたはコードセグメントを検索します。
3	[+]	クリックして、ソリューション、ビジネスフロー、フォルダ、ノード、テーブル、リソース、または関数エントリを作成します。
4	**Reload**	クリックして、現在のディレクトリツリーを更新します。
5	**Locate**	クリックして、選択したファイルの場所を特定します。
6	**Import**	クリックして、ローカルデータをオンラインテーブルへインポートします。 エンコード形式に注意します。
7	**Filter**	クリックして、指定した条件に基づいてノードをフィルタリングします。
8	**Save**	クリックして、現在のコードを保存します。
9	**Save as Query File**	クリックして、現在のコードを一時ファイルとして保存します。このファイルは**[Temporary query]** 列に表示されます。
10	**Submit**	クリックして、現在のノードを送信します。

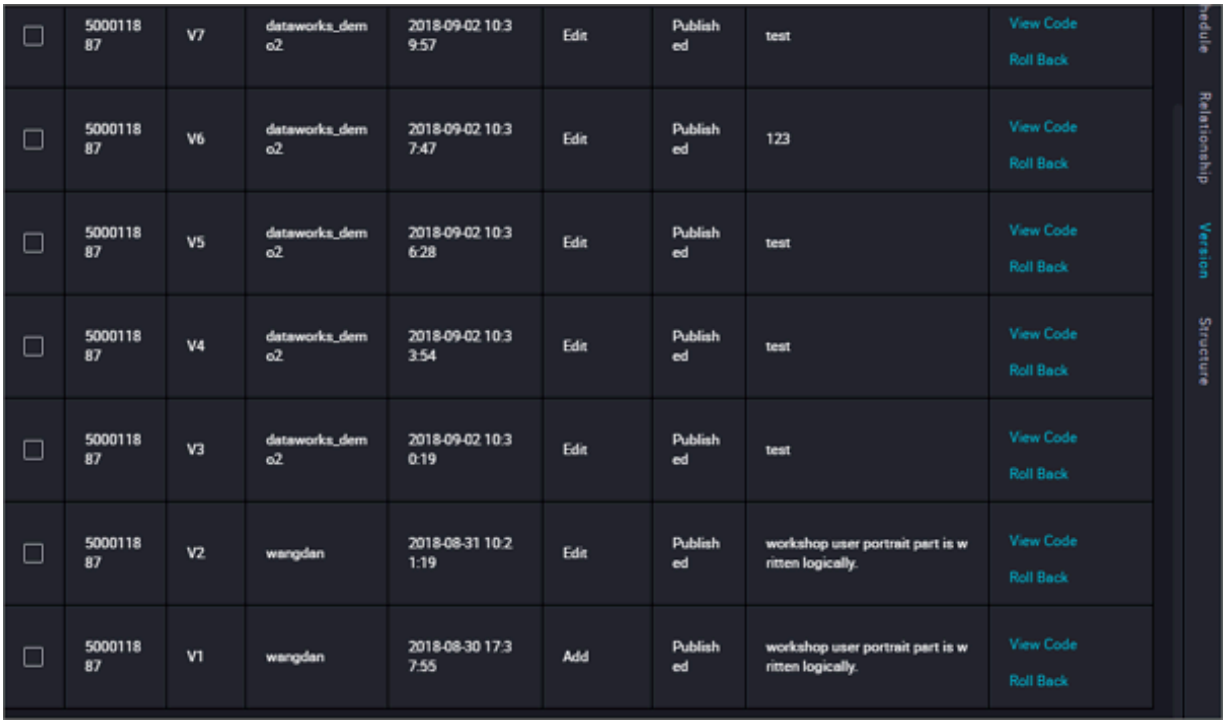
No.	機能	説明
11	Submit and Unlock	クリックして、現在のノードを送信し、コードを編集できるようにロックを解除します。
12	Steallock	このコードのオーナーではない場合に、クリックしてノードを編集します。
13	Run	クリックして、現在のノードのコードを実行します。
14	Run After Setting Parameters	クリックして、設定したパラメーターを使って現在のノードのコードを実行します。
15	Precompile	クリックして、現在のノードのパラメーターを編集およびテストします。
16	Stop Run	クリックして、現在実行中にコードを停止します。
17	Reload	クリックして、ページを更新し、以前保存したページへ戻ります。
18	Run Smoke Test in Development Environment	クリックして、開発環境にある現在のノードのコードをテストします。
19	View Smoke Test Log in Development Environment	クリックして、開発環境で実行されているノードの実行ログを表示します。
20	Go to Scheduling System of Development Environment	クリックして、開発環境の O&M センターへ移動します。
21	Format	クリックして、現在のノードのコードを配列します。コードを1 行におさめるには長すぎる場合に使用します。
22	Publish	クリックして、送信したコードを公開します。 コードが公開された後、コードは運用環境にあります。
23	O&M	クリックして、運用環境の O&M センターへ移動します。
24	Scheduling Configuration	クリックして、ノードのスケジューリング属性、パラメーター、およびリソースグループを設定します。
25	Relationship	クリックして、コードで使用されるテーブルの関係を表示します。
26	Version	クリックして、現在のノードの送信レコードと公開レコードを表示します。

No.	機能	説明
27	**Structure**	クリックして、現在のノードのコード構造を表示します。 コードが長すぎる場合、構造のキー情報を元にコードセグメントを簡単に検索することができます。

1.3.2 バージョン

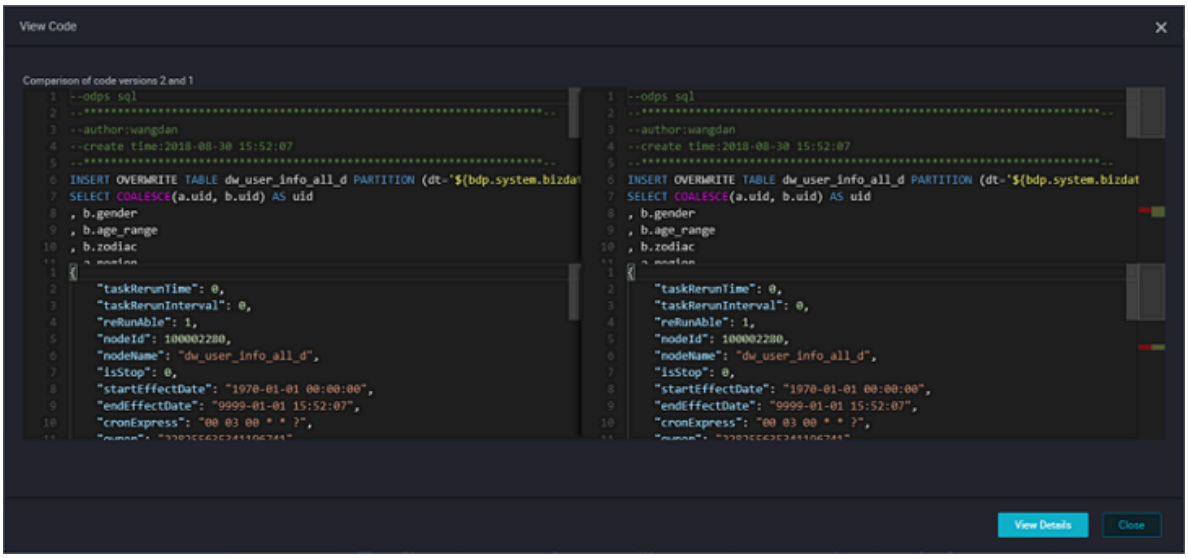
バージョンとは、現在のノード送信レコードとリリースレコードです。送信を行うたびに新しいバージョンが生成されます。 ノードの操作を簡単にするために、必要に応じて関連ステータスの確認、タイプの変更、備考のリリースを行うことができます。

注：
送信されたノードのみにバージョン情報が存在します。



- **File ID:** 現在のノード **ID**
- **Version:** リリースするたびに新しいバージョンが生成されます。 **1** 回目のリリースは **V1**、**2** 回目の編集では **V2** などです。
- **Submitter:** ノードの送信やリリース操作を行ったユーザー
- **Submission Time:** バージョンのリリース時間 バージョンが送信後にリリースされた場合、リリース時間が送信時間となります。 デフォルトでは、操作の最終リリース時間が記録されます。
- **Change Type:** 現在のノードの操作履歴 ノードの**1** 回目のリリースの場合、**[Added]** と設定されます。ノードが編集された場合は、**[Modified]** と設定されます。

- **Status:** 現在のノードの操作ステータスレコードです。
- **Remarks:** 送信される際の現在のノードの説明を変更します。 ノードを操作中に他の担当者が関連したバージョンを探す際に役立ちます。
- **Action:** この列では、**[Code]** と **[Roll Back]** を選択できます。
  - **View Code:** クリックしてバージョンコードを表示し、元に戻すレコードバージョンを正確に検索します。
  - **Roll Back:** クリックして現在のノードを必要に応じて以前のバージョンへ元に戻します。元に戻した後、ノードをリリースするために再度送信する必要があります。
- **Compare:** クリックして 2 つのバージョンのコードとパラメーターを比較します。



[View Details] をクリックし、詳細ページへ移動します。コードとスケジューリング属性の変更を比較します。

> 注：
> 2 つのバージョンでのみ比較することができます。 3 つ以上のノードは比較できません。

1.3.3 構造

構造は現行コードに基づいており、**SQL** で実行されるプロセス図が記述されます。編集した **SQL** を即座に確認できるため、編集および表示を簡単に行えます。

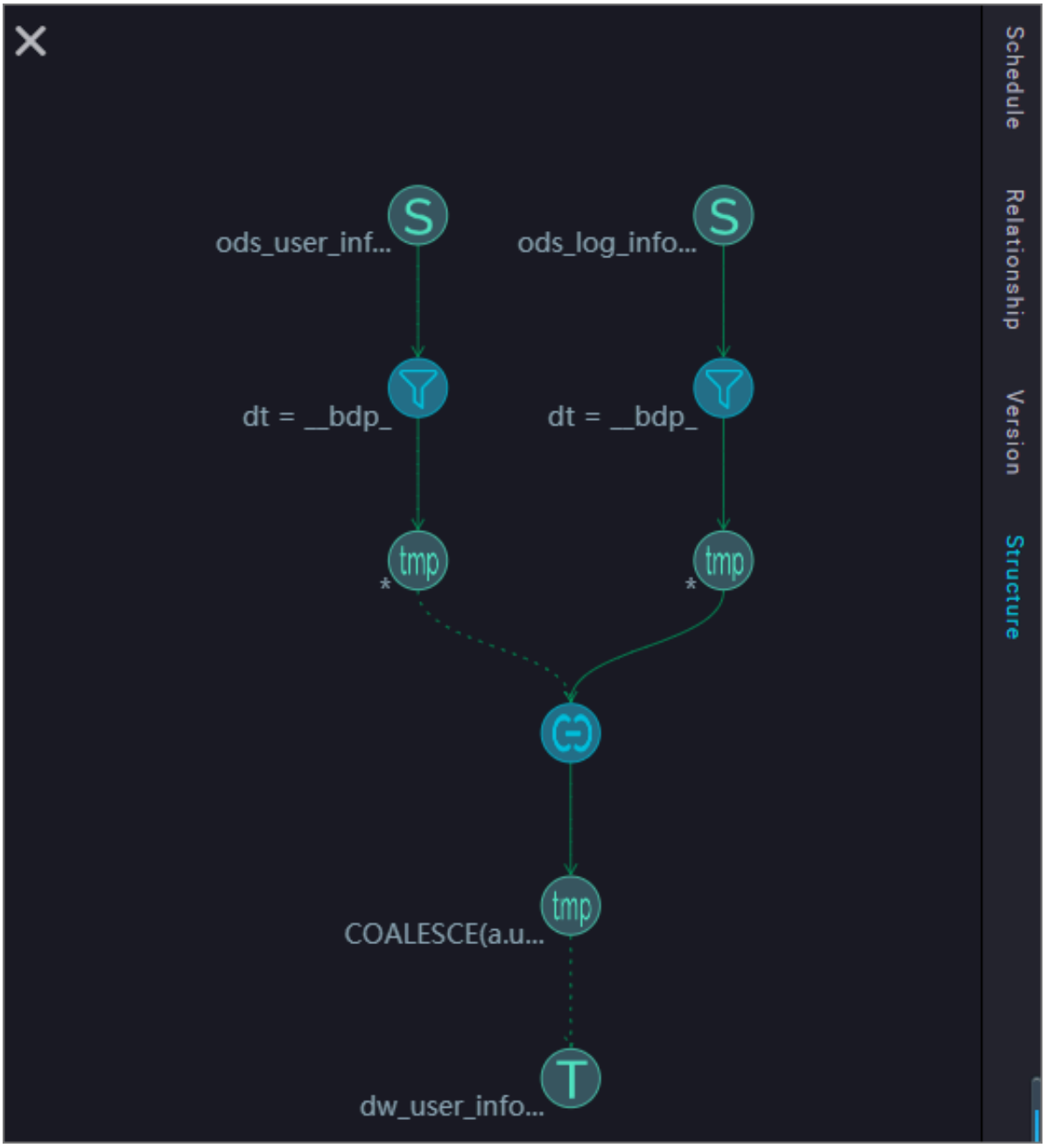
構造

SQL は次のとおりです。

```
INSERT OVERWRITE TABLE dw_user_info_all_d PARTITION (dt='${bdp.system.
bizdate}')
SELECT COALESCE(a.uid, b.uid) AS uid
  , b.gender
  , b.age_range
```

```
  , B. flavdiac
  , a.region
  , a.device
  , a.identity
  , a.method
  , a.url
  , a.referer
  , a.time
FROM (
  VALUES
  From fig
  WHERE dt = ${bdp.system.bizdate}
) a
LEFT OUTER JOIN (
  VALUES
  FROM ods_user_info_d
  WHERE dt = ${bdp.system.bizdate}
) b
on a.uid = b.uid ;
```

このコードによると、構造は次のように説明されます。



円にマウスを置くと、対応する説明が表示されます。

1. **Source table: SELECT** クエリのターゲットテーブルです。
2. **Filter:** 照会するテーブルの特定のパーティションにフィルターをかけます。
3. 中間のテーブル (クエリビュー) の **1** 番目の部分**:** 一時テーブルへクエリデータの結果を配置します。
4. **Join:** 結合機能を使って **2** 部クエリの結果をモザイクします。
5. **2** 番目のセクションにある 中間テーブル (クエリビュー)**:** 結合結果を一時テーブルへ要約します。この一時テーブルは、**3** 日保管され、**3** 日後に自動的にクリアされます。
6. **Target table (insert): 2** 番目で取得されたデータをテーブルへ挿入し上書きします。

1.3.4 リレーションシップ

類似リレーションは、現在のノードとその他のノードのリレーションシップを示します。 リレーションシップには 2 つの部分、依存ダイアグラムと内部リレーションシップマップがあります。

依存グラフ

ノードの依存性によって、依存グラフは現在のノードが想定しているものかどうかを示します。想定しているものではない場合、**Schedule** 設定インターフェイスへ戻りリセットします。

内部リレーションシップマップ

内部リレーションシップマップは、ノードのコードに基づいて説明します。たとえば

```
INSERT OVERWRITE TABLE dw_user_info_all_d PARTITION (dt='${bdp.system.
bizdate}')
SELECT COALESCE(a.uid, b.uid) AS uid
   , b.gender
   , b.age_range
   , B. flavdiac
   , a.region
   , a.device
   , a.identity
   , a.method
   , a.url
   , a.referer
   , a.time
FROM (
  VALUES
  From fig
  WHERE dt = ${bdp.system.bizdate}
) a
LEFT OUTER JOIN (
  VALUES
  FROM ods_user_info_d
  WHERE dt = ${bdp.system.bizdate}
) b
```

```
on a.uid = b.uid ;
```

SQL によると、次の内部リレーションシップマップは解決され、テーブル間のリレーションシップを示すために結合モザイクとして使用されるアウトプットテーブルを説明します。

1.4 ノードタイプ

1.4.1 ノードタイプの概要

DataWorks では、7 つのタイプのノードが提供されます。異なる使用ケースが適用されます。

仮想ノードタスク

仮想ノードは、データを生成しない制御ノードです。 通常、ワークフローでのノードの全体的な計画に対して、ルートノードを使用します。 仮想ノードタスクについての詳細は、「仮想ノード」をご参照ください。

> 注：
> ワークフローの最終出力テーブルには、複数の分岐入力テーブルが含まれます。 仮想ノードは通常、これらテーブルに依存関係がない場合に使用されます。

ODPS SQL タスク

ODPS SQL タスクでは、**Web** 上で **SQL** コードを直接編集したり管理することができます。また、簡単に実装の実行やデバッグ、共同開発を行うことができます。 **DataWorkd** では、コードバージョンの管理、上位および下位の依存関係の自動解決などの機能が提供されます。 例の詳細については、「*ODPS SQL ノード*」をご参照ください。

DataWorks では、デフォルトで **MaxCompute** のプロジェクトが開発および運用スペースとして使用されます。このため、**ODPS SQL** ノードのコードは、**MaxCompute SQL** 構文に従います。 **MaxCompute SQL** 構文は、**Hive** 構文のように標準の **SQL** のサブセットとして使用できます。 しかし、**MaxCompute SQL** はデータベースと同一視することはできません。これは、トランザクション、主キーの制約、インデックスといったデータベースにある機能の多くが、**MaxCompute SQL** にはないためです。

特定の**MaxCompute SQL** 構文についての詳細は、「*SQL の概要* 」をご参照ください。

ODPS MR タスク

MaxCompute では、**MapReduce** プログラミング **API** がサポートされています。これらの **Java API** は、**MaxCompute** のデータプロセスで **MapReduce** プログラムをコンパイルするために使用できます。 **ODPS MR** ノードを作成しタスクスケジューリングに使用することができます。 例の詳細については、「*ODPS MR ノード*」をご参照ください。

PyODPS タスク

MaxCompute では *Python SDK* が提供されます。これを使って **MaxCompute** を操作します。

DataWorks では、**PyODPS** タスクタイプが提供され、 **MaxCompute** の **Python SDK** を統合することができます。 **Python** コードを直接編集し、**DataWorks** の **PyODPS** ノードの **MaxCompute** を操作することができます。 詳細は、「*PyODPS ノード*」をご参照ください。

SQL コンポーネントノード

SQL コンポーネントノードは、**SQL** コードプロセステンプレートで、複数の入出力パラメーターが含まれます。 **SQL** コードプロセスを操作するには、**1** つ以上のソースデータテーブルをインポートしフィルターをかけます。結合して集約し、新しいビジネスに必要なターゲットテーブルを形成します。 詳細は、「*SQL コンポーネントノード*」をご参照ください。

データ同期タスク

データ同期ノードタスクは、**Alibaba Cloud DTplus** プラットフォームで提供される安定性と効率性に優れ、また自動的でスケーラブルな外部データ同期クラウドサービスです。 データ同期ノードを使うと、ビジネスシステムのデータを **MaxCompute** へ簡単に同期することができます。 詳細は、「データ統合ノード」をご参照ください。

1.4.2 データ統合ノード

現在、データ統合タスクでサポートされているデータソースは、**MySQL**、 **DRDS**、**SQL Server**、**PostgreSQL**、 **Oracle**、 **MongoDB**、 **DB2**、 **OTS**、 **OTS Stream**、**OSS**、**FTP**、

Hbase、**LogHub**、 **HDFS**、および **Stream** です。 サポートされているデータソースについての詳細は、「*#unique_17*」をご参照ください。

統合タスクの設定

詳細は、次をご参照ください。 *#unique_18/unique_18_Connect_42_section_tfn_1kc_p2b*

ノードスケジューリングの設定

ノードタスクの編集エリアの右側にある **[Scheduling Configuration]** をクリックし、**[node scheduling configuration]** ページへ移動します。 詳細は、「スケジューリング設定」をご参照ください。

ノードの送信

設定が完了した後、ページの左上隅にある **[Save]** をクリックするか、または **[Ctrl]** + **[S]** を押してノードを開発環境へ送信 (ロックを解除) します。

ノードタスクの公開

操作の詳細については、「リリース管理」をご参照ください。

本番環境でのテスト

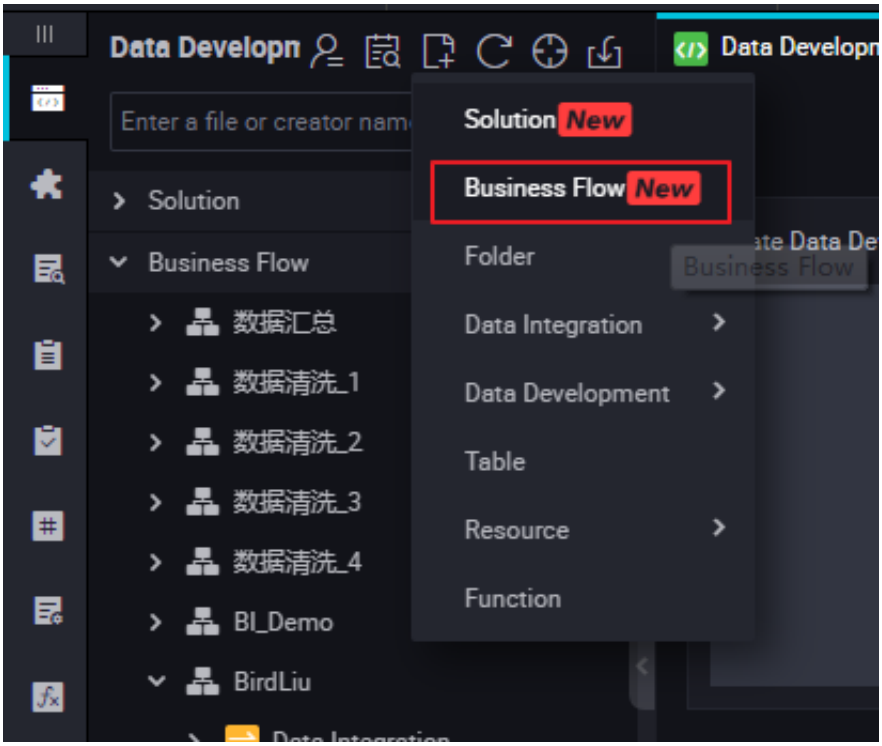
詳細は、「*#unique_20*」をご参照ください。

1.4.3 ODPS SQL ノード

ODPS SQL 構文は、**SQL** 構文と同じです。データは大容量 (**TB** 級) だが、リアルタイム要件が高くない分散シナリオに適用されます。 スループット向けの **OLAP** アプリケーションです。ジョブの準備から送信までのプロセスが完了するのに長時間かかるため、ビジネスで数万件のトランザクション処理が必要な場合に **ODPS SQL** の使用を推奨します。

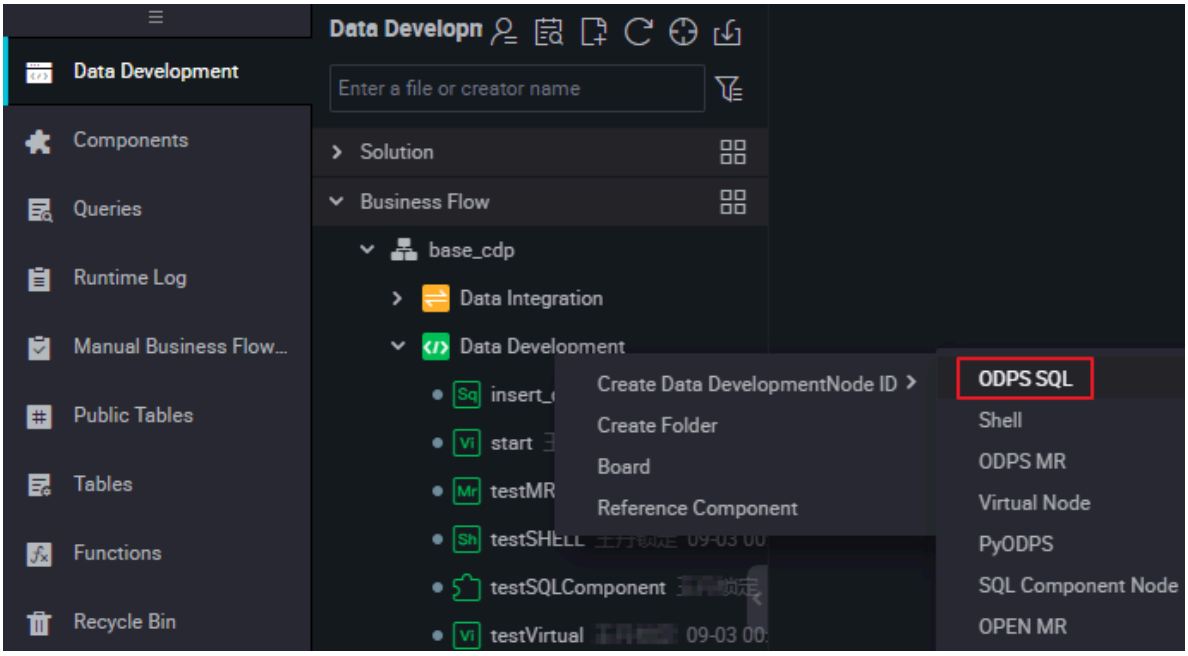
1. ビジネスフローを作成します。

   [Data Development] の **[Business Flow]** を右クリックし、**[Create Business Flow]** を選択します。



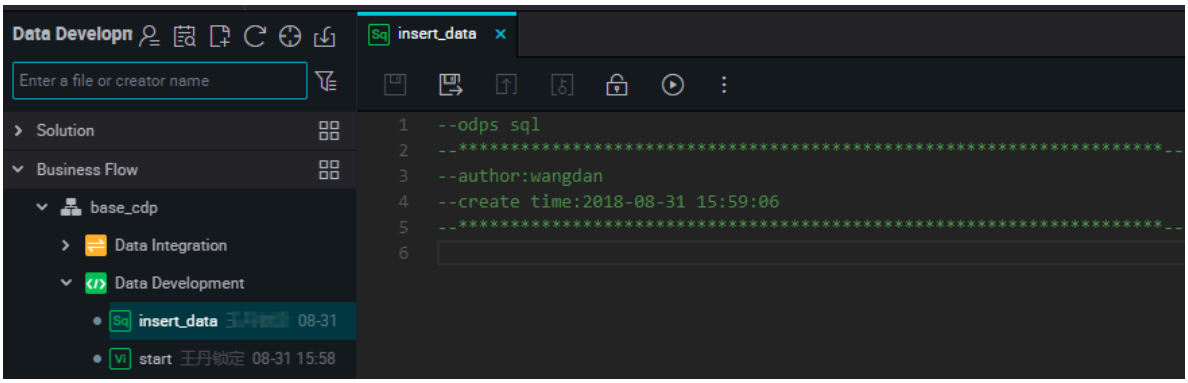
2. **ODPS SQL** ノードを作成します。

   [Data Development] を右クリックし、**[Create Data Development Node]** > **[ODPS SQL]** の順に選択します。



3. ノードコードを編集します。

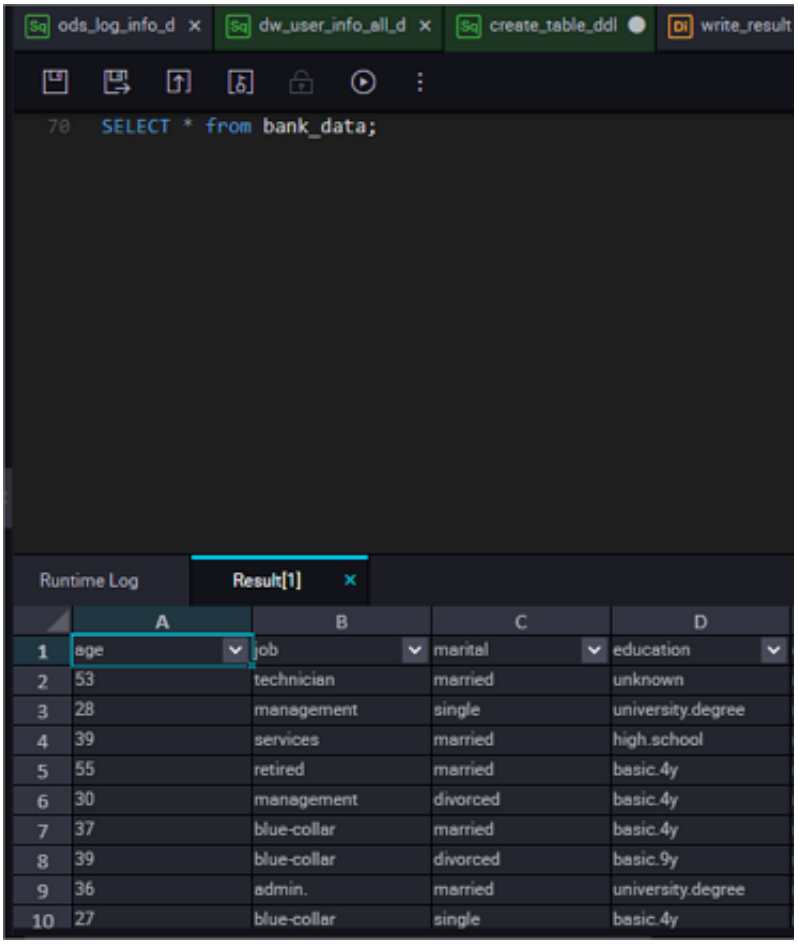
   SQL 文の構文についての詳細は、「*MaxCompute SQL* 文」 をご参照ください。



4. クエリ結果の表示

DataWorks のクエリ結果は、スプレッドシート機能に連携されるため、データ結果を簡単に操作できます。

クエリ結果は、直接スプレッドシート形式で表示されます。 **DataWorks** で操作を行ったり、スプレッドシートを開いたり、ローカルのエクセルへコンテンツステーションを自由にコピーできます。



	A	B	C	D
1	age	job	marital	education
2	53	technician	married	unknown
3	28	management	single	university.degree
4	39	services	married	high.school
5	55	retired	married	basic.4y
6	30	management	divorced	basic.4y
7	37	blue-collar	married	basic.4y
8	39	blue-collar	divorced	basic.9y
9	36	admin.	married	university.degree
10	27	blue-collar	single	basic.4y

- **Hidden column:** 1 つ以上の非表示になっている列を選択し、列を非表示にします。
- **Copy the row:** コピーする 1 つ以上の行の左側を選択し、**[Copy the row]** をクリックします。
- **Copy the column:** 上部の列で 1 つ以上の列またはポイントを選択し列をコピーします。
- **Copy:** 選択したコンテンツを自由にコピーします。
- **Search:** クエリ結果の右上隅に検索ボックスが表示されます。これを使ってテーブル内のデータを簡単に検索できます。

5. ノードスケジューリングの設定

   ノードタスク編集エリアの右側にある **[Schedule]** をクリックし、**[node scheduling configuration]** ページへ移動します。 詳細は、「スケジューリングの設定」をご参照ください。

6. ノードを送信します。

   設定が完了した後、ページの左上隅にある **[Save]** をクリックするか、または **[Ctrl]** + **[S]** を押してノードを開発環境へ送信 (およびロックの解除) します。

7. ノードタスクを発行します。

   操作に関する詳細は、「リリースの管理」をご参照ください。

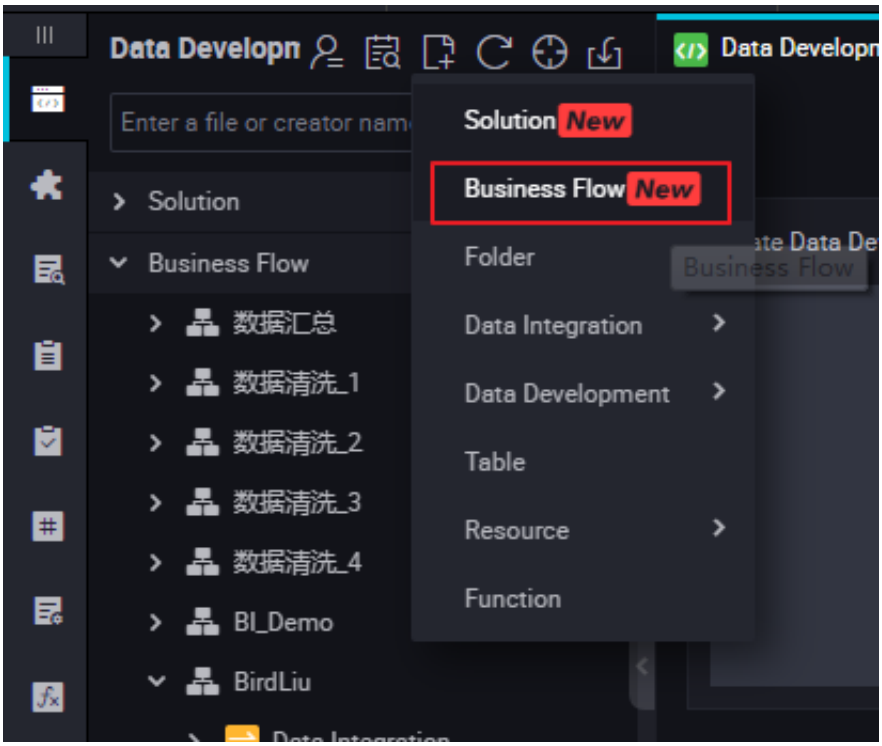
8. 本番環境でテストします。

   操作に関する詳細は、「*#unique_20*」をご参照ください。

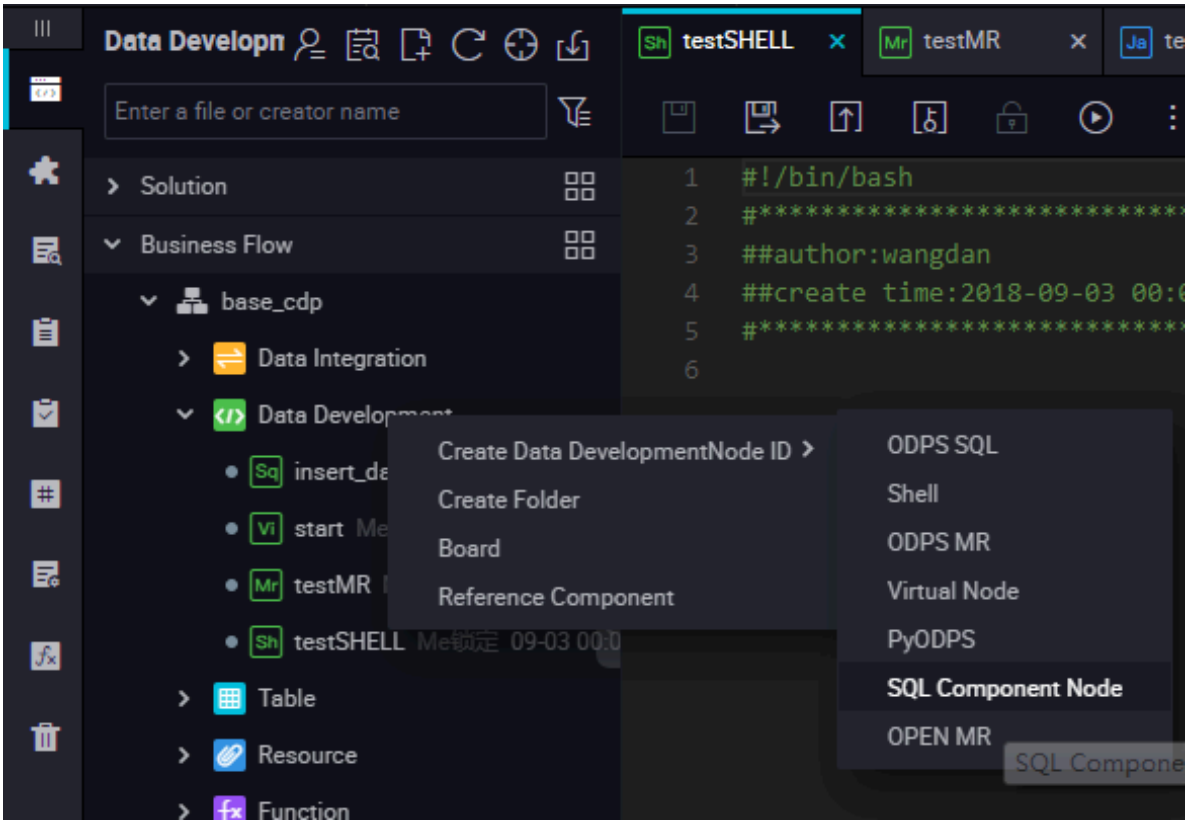
1.4.4 SQL コンポーネントノード

手順

1. **[Data Development]** の **[Business Flow]** を右クリックし、**[Create Business Flow]** を選択します。



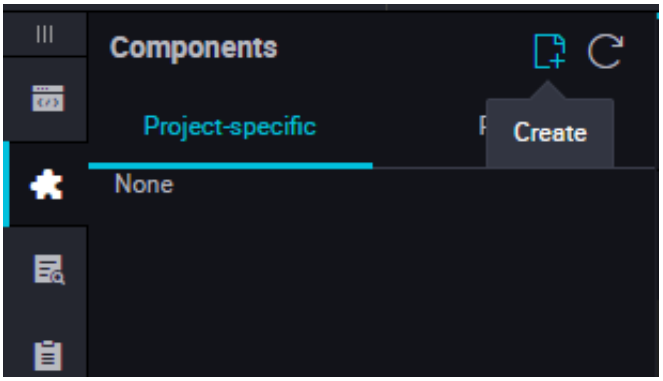
2. **[Data Development]** を右クリックし、**[Create Data Development Node]** > **[SQL Component Node]** の順に選択します。



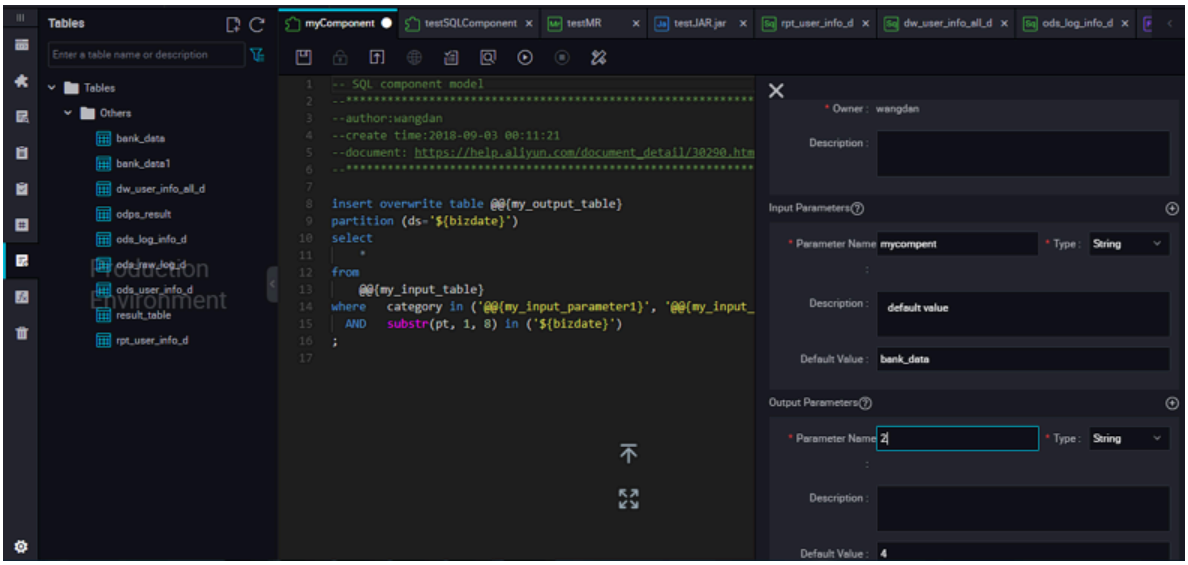
3. 開発の効率を向上させるため、データタスクの開発者はプロジェクトメンバーとテナントメンバーによって提供されるコンポーネントを使用して、データプロセスノードを作成することができます。

   - ローカルプロジェクトのメンバーによって作成されたコンポーネントは、**Project Components** の下にあります。
   - テナントメンバーによって作成されたコンポーネントは、**Public Components** の下にあります。

   ノードを作成する際は、ノードタイプを 「**SQL Component node**」 タイプへ設定し、ノード名を指定します。



選択したコンポーネントのパラメーターを指定します。



パラメーター名を入力し、パラメータータイプを「**Table**」または「**String**」に設定します。

3 つの **get_top_n** パラメーターを順番に指定します。

Table タイプのパラメーターに対する入力テーブル**"test_project.test_table'** を指定します

4. ノードスケジューリングの設定

   ノードタスク編集エリアの右側で **[Scheduling Configuration]** をクリックし、**[node scheduling configuration]** ページへ移動します。 詳細は、「スケジューリングの設定」 をご参照ください。

5. ノードを送信します。

   設定が完了した後、ページの左上隅にある **[Save]** をクリックするか、または **[Ctrl]** + **[S]** を押してノードを開発環境へ送信 (およびロックの解除) します。

6. ノードを発行します。

   操作の詳細については、「リリースの管理」をご参照ください。

7. 本番環境でテストします。

   操作の詳細については、「*#unique_20*」をご参照ください。

SQL コンポーネントノードのバージョンのアップグレード

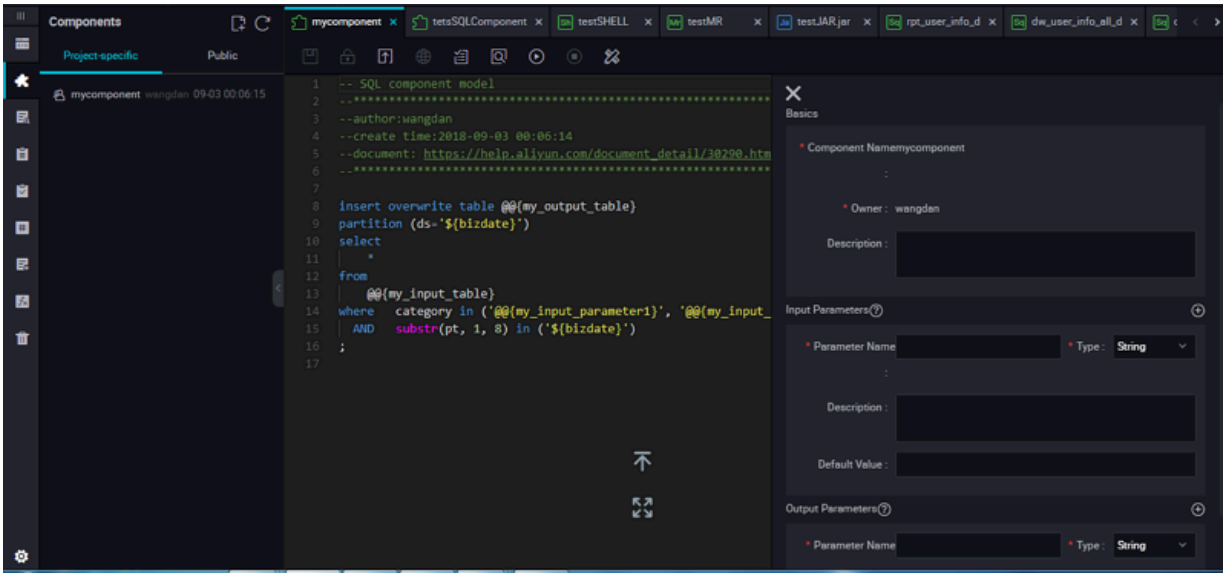
コンポーネント開発者が新しいバージョンをリリースした後、コンポーネントユーザーは、既存のコンポーネントのインスタンスを最新版のコンポーネントへアップグレードするかどうか選択することができます。

コンポーネントのバージョンメカニズムにより、開発者はコンポーネントを継続してアップグレードすることができるため、アップグレード後ユーザーは向上したプロセス実行効率や最適化されたビジネス効果を享受することができます。

たとえば、ユーザー **A** が ユーザー **C** が開発した **v1.0** コンポーネントを使用しており、コンポーネントオーナーの **C** はコンポーネントを **V.2.0** へアップグレードします。 アップグレード後、ユーザー **A** は **v1.0** コンポーネントを継続して使用できますが、アップグレードリマインダーを受け取ります。 旧コードと新コードを比較した後、ユーザー **A** が新しいバージョンのビジネス効果の方が旧バージョンよりも良いと思った場合は、コンポーネントを最新版へアップグレードするかどうかを判断することができます。

コンポーネントテンプレートに基づいて開発された **SQL** コンポーネントノードをアップグレードする場合、**[Upgrade]** を選択し、**SQL** コンポーネントノードが新しいバージョンでも有効かどうかパラメーターの設定で確認します。新しいバージョンのコンポーネントの指示に従って必要な調整を行い、通常の **SQL** コンポーネントノードと同じようにノードを送信しリリースします。

インターフェイスの機能



インターフェイスの機能は次のとおりです。

No.	機能	説明
1	**Save**	クリックして、現在のコンポーネントの設定を保存します。
2	**Steal lock Edit**	現在のコンポーネントのオーナーでない場合、クリックして、ロックを解除してノードを編集することができます。
3	**Submit**	クリックして、現在のコンポーネントを開発環境へ送信します。
4	**Publish Component**	クリックして、すべてのテナントに対しユニバーサルグローバルコンポーネントを発行します。これによりテナント内のすべてのユーザーがパブリックコメントを表示し使用することができます。
5	**Resolve Input and Output Parameters**	クリックして、現在のコードの出入力パラメーターを解決します。
6	**Precompilation**	クリックして、現在のコンポーネントのカスタムパラメーターおよびコンポーネントパラメーターを編集します。
7	**Run**	クリックして、開発環境内でコンポーネントをローカル上で実行します。
8	**Stop Run**	クリックして、実行中のコンポーネントを停止します。
9	**Format**	クリックして、キーワード別に現在のコンポーネントコードをソートします。

No.	機能	説明
10	**Parameter Settings**	クリックして、コンポーネントの情報、入力パラメーターの設定、出力パラメーターの設定を表示します。
11	**Version**	クリックして、現在のコンポーネントの送信およびリリースレコードを表示します。
12	**Reference Records**	クリックして、コンポーネントの使用レコードを表示します。

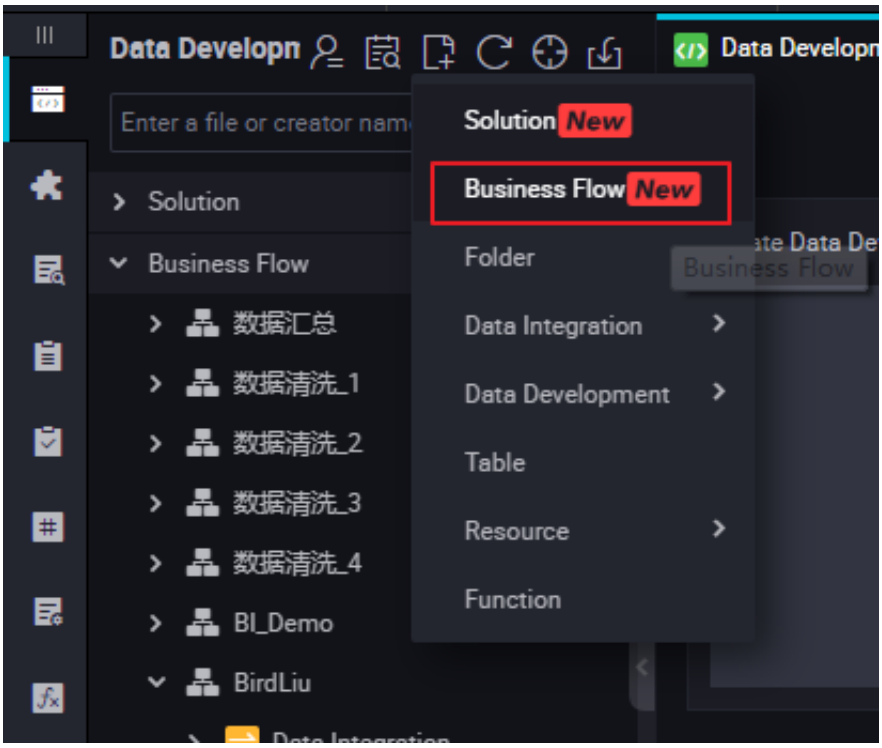
1.4.5 仮想ノード

仮想ノードは、データを生成しない制御ノードです。 通常、ワークフローのノードの全体的な計画に対するルートノードとして使用されます。

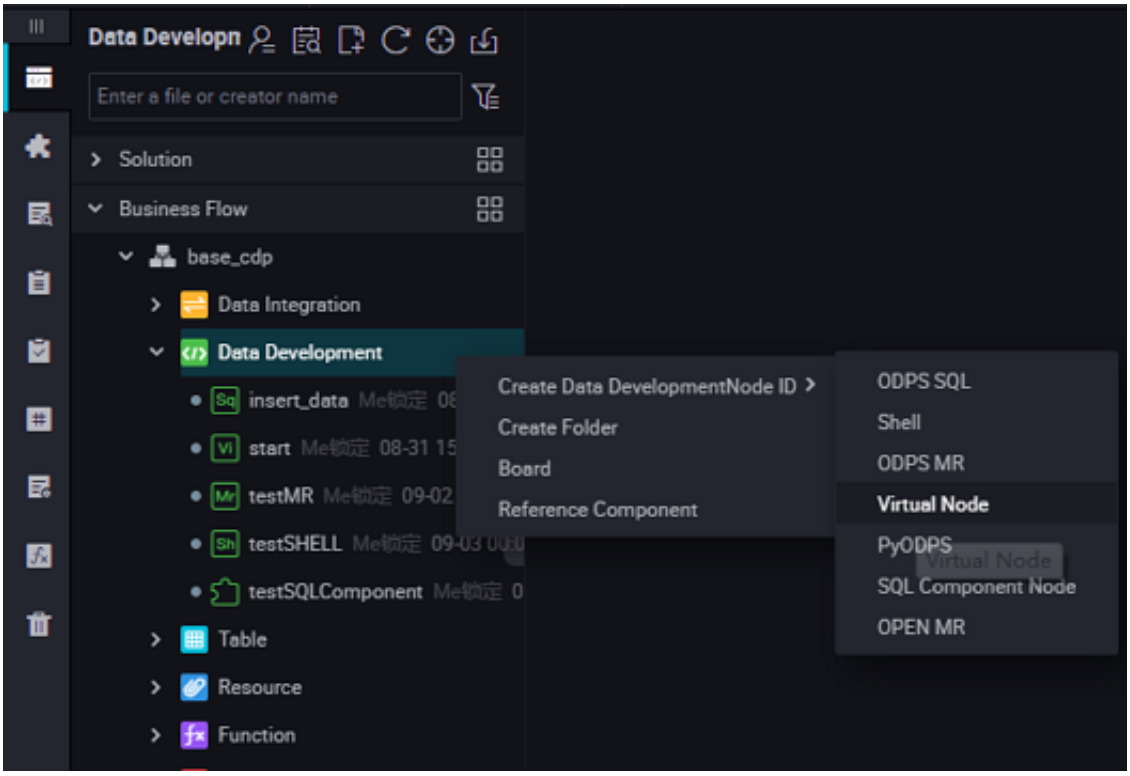
注：
ワークフローの最終出力テーブルには、複数の分岐入力テーブルが含まれます。 仮想ノードは通常、これら入力テーブルに依存関係がない場合に使用されます。

仮想ノードタスクの作成

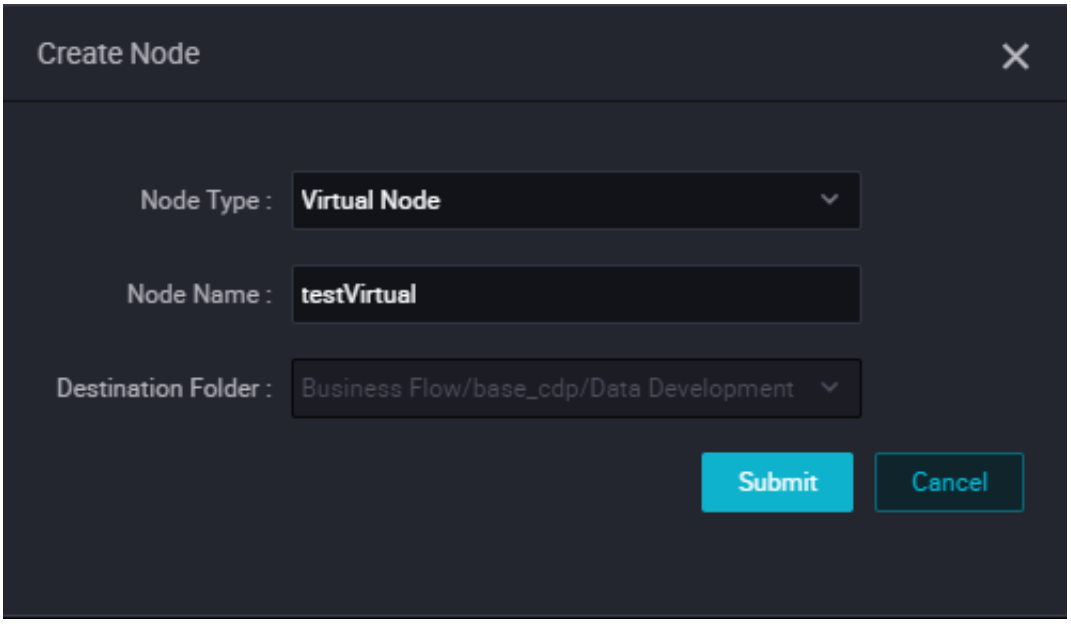
1. **[Data Development]** で **[Business Flow]** を右クリックし、**[Create Business Flow]** を選択します。



2. **[Data Development]** を右クリックし、**[Create Data Development Node]** > **[Virtual Node]** の順に選択します。



3. ノードタイプを **[Virtual Node]** に設定し、ノード名を入力します。ターゲットフォルダーを選択し、**[Submit]** をクリックします。



4. ノードコードの編集**:** 仮想ノードのコードを編集する必要はありません。

5. ノードスケジューリングの設定

   ノードタスク編集エリアの右側にある **[Schedule]** をクリックし、**[node scheduling configuration]** ページへ移動します。 詳細は、「スケジューリングの設定」 をご参照ください。

6. ノードを送信します。

   設定が完了した後、ページの左上隅にある **[Save]** をクリックするか、 または **[Ctrl]** + **[S]** を押してノードを開発環境へ送信 (およびロックの解除) します。

7. ノードタスクを発行します。

   操作の詳細については、「リリースの管理」をご参照ください。

8. 本番環境でテストします。

   操作の詳細については、「*#unique_20*」をご参照ください。

1.4.6 ODPS MR ノード

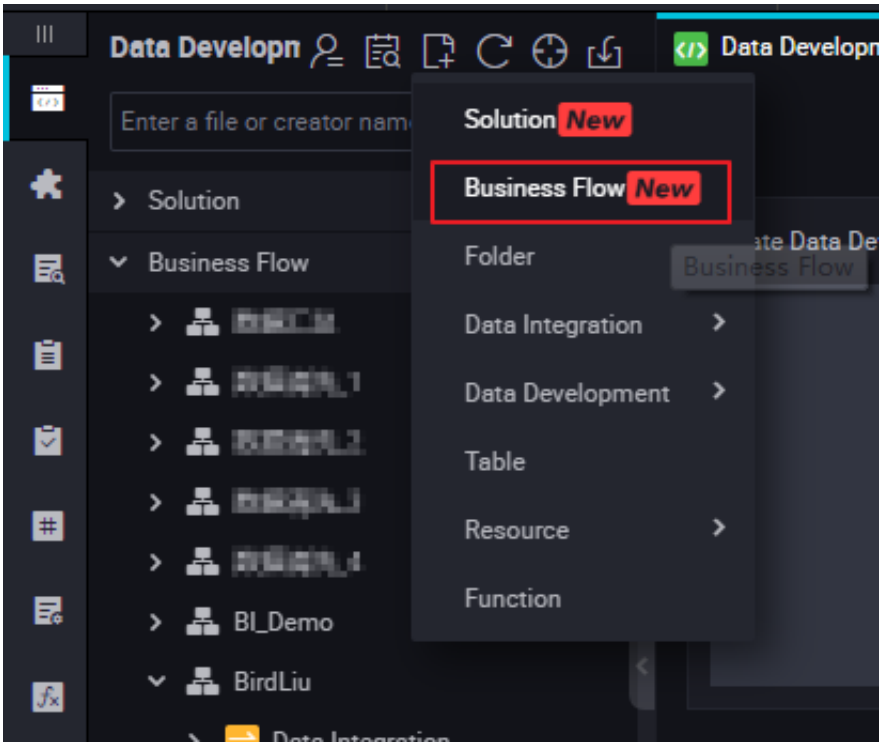
MaxCompute では、**MapReduce** プログラミング **API** を使用できます。 **MaxCompute** でデータを処理するためには、**MapReduce** によって提供される **Java API** を使って、**MapReduce** プログラムを記述します。 **ODPS MR** ノードを作成し、タスクスケジューリングで使用することができます。

DPS MR の編集方法や使用方法については、**MaxCompute** ドキュメントの「*WordCount* 例」にある例をご参照ください。

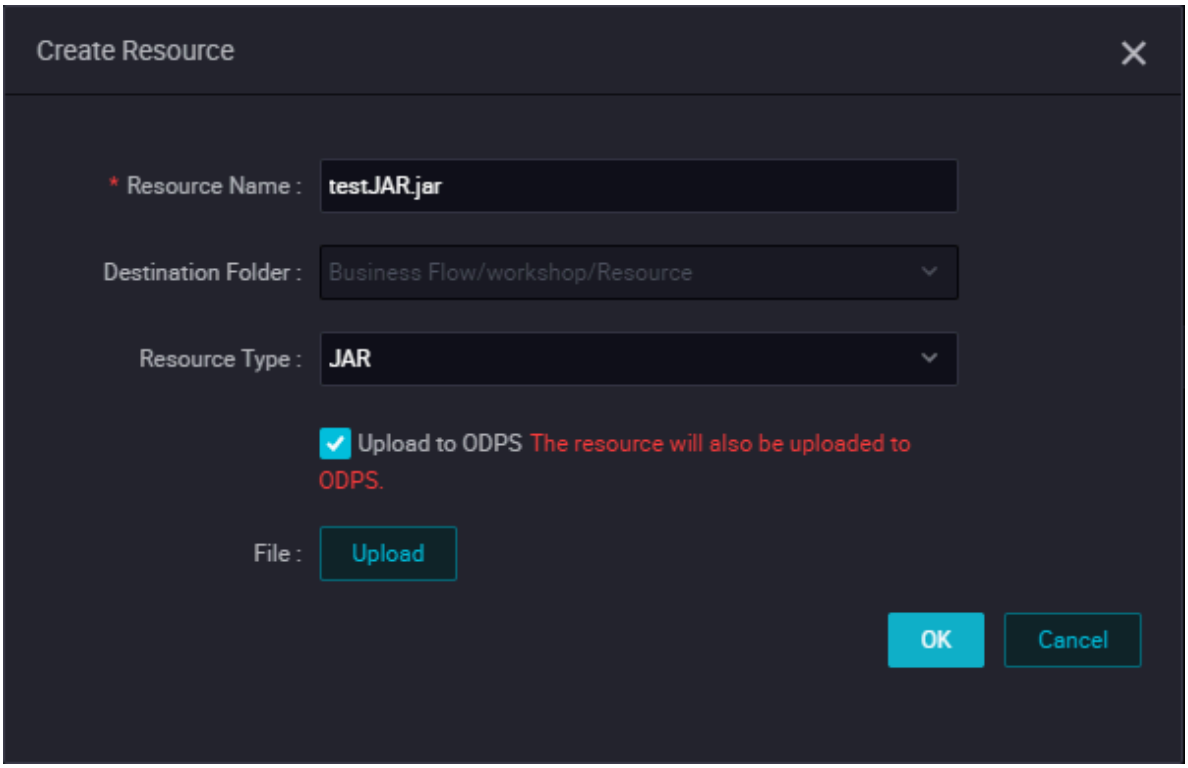
ODPS MR ノードを使用するには、使用するリソースをアップロードし、リリースします。その後、**ODPS MR** ノードを作成します。

リソースインスタンスの作成

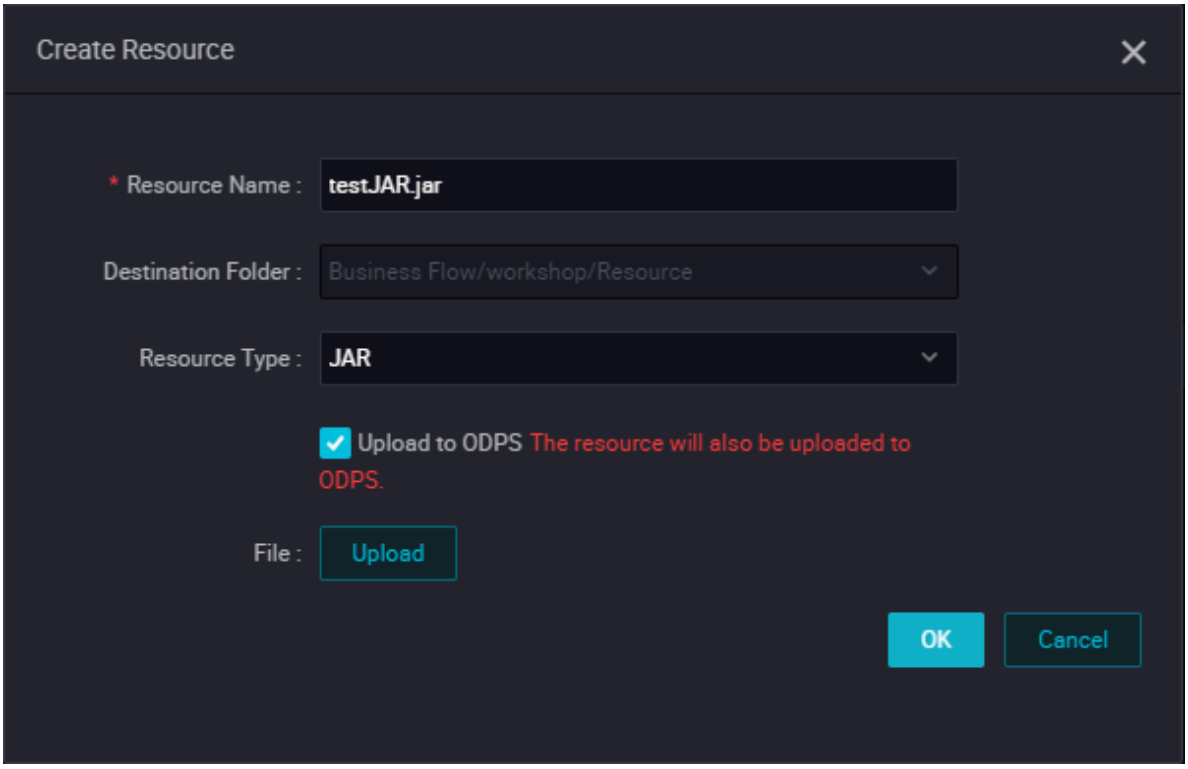
1. **[Data Development]** の **[Business Flow]** を右クリックし、**[Create Business Flow]** を選択します。



2. **[Resource]** を右クリックし、**[Create Resource]** > **[jar]** の順に選択します。



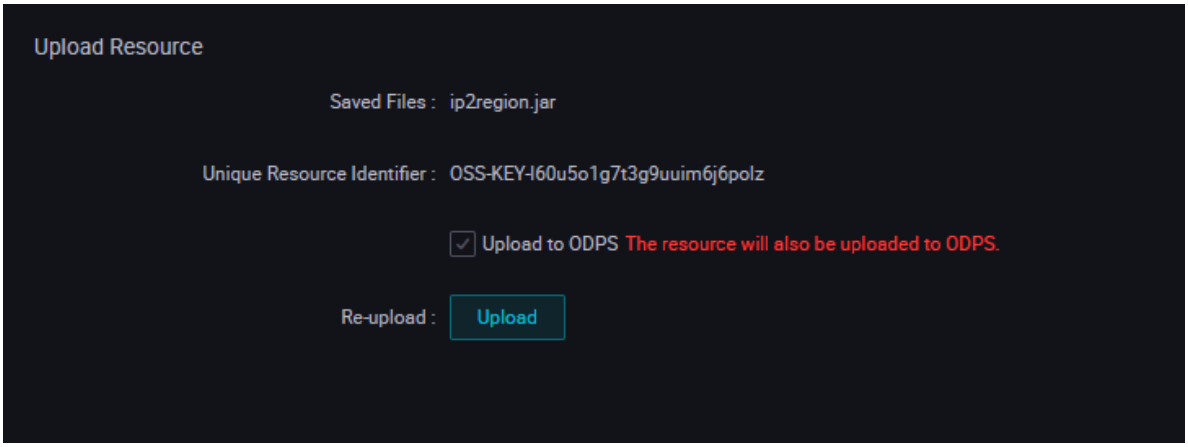
3. 命名規則に従って、**[Create Resource]** にリソース名を入力します。リソースタイプを「**jar**」に設定し、ローカルの **jar** パッケージを選択します。



注：

- **jar** パッケージが **ODPS** クライアントへアップロードされた後、**[Uploaded to ODPS]** の選択を解除します。 解除をしないと、アップロードプロセス中にエラーが報告されます。
- リソース名は、アップロードファイルと同じ名前にする必要はありません。
- リソース名の命名規則**: 1** 文字以上 **128** 文字以内の文字列で、文字、数字、アンダースコア、ドットを使用できます。 リソース名は大文字と小文字が区別されません。 リソースが **jar** リソースの場合、拡張子は **.jar** です。 リソースが **Python** リソースの場合、拡張子は **.py** です。

4. **[Submit]** をクリックし、リソースを開発スケジューリングサーバーへ送信します。

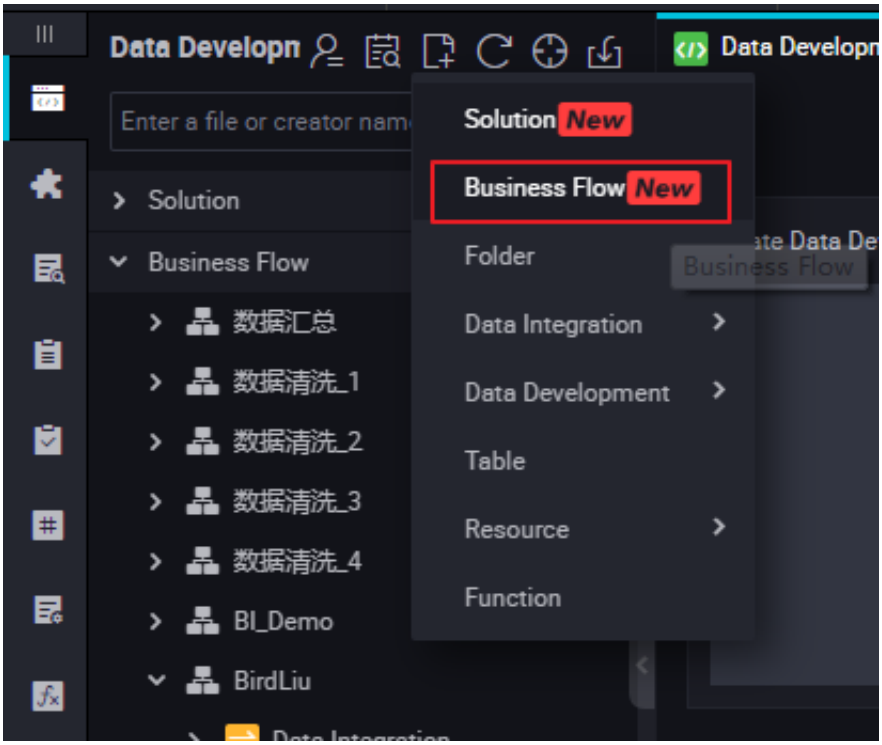


5. ノードタスクを発行します。

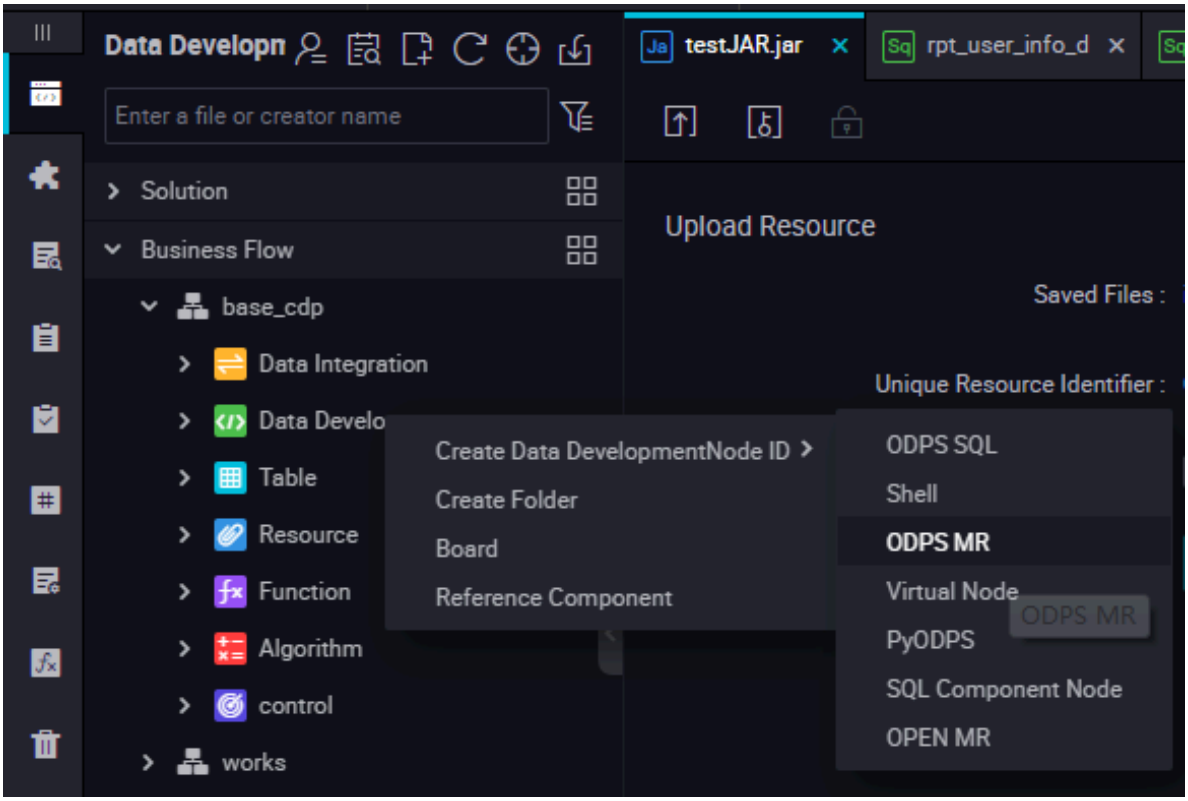
   操作の詳細については、「リリースの管理」をご参照ください。

ODPS MR ノードの作成

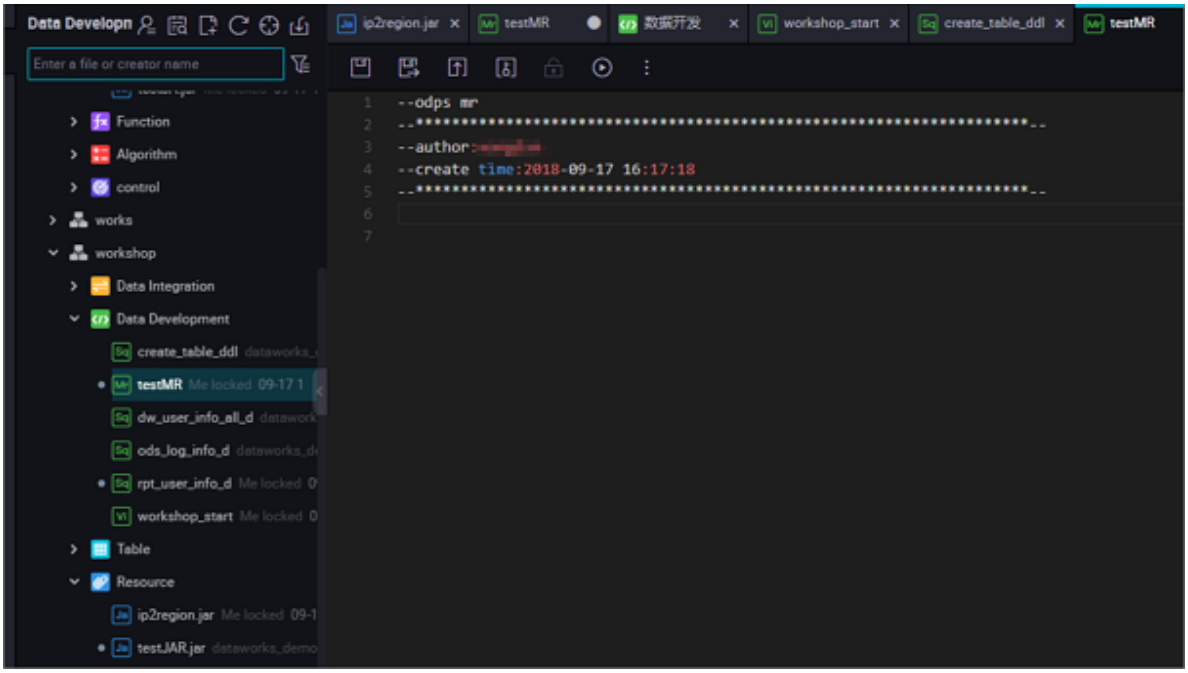
1. **[Data Development]** の **[Business Flow]** を右クリックし、**[Create Business Flow]** を選択します。



2. **[Data Development]** を右クリックし、**[Create Data Development Node]** > **[ODPS MR]** の順に選択します。



3. ノードコードを編集します。 新規の **ODPS MR** ノードをダブルクリックし、次のインターフェイスへ移動します。



ノードコードの編集例:

```
jar -resources base_test.jar -classpath ./base_test.jar com.taobao.
edp.odps.brandnormalize.Word.NormalizeWordAll
```

コードの説明は次のとおりです。

- `-resources base_test.jar`: 参照される **jar** リソースのファイル名を示します。
- `-classpath`: **jar** パッケージのパスです。
- `com.taobao.edp.odps.brandnormalize.Word.NormalizeWordAll`: 実行中に呼び出される **jar** パッケージのメインクラスを示します。 **jar** パッケージのメインクラス名と同じにする必要があります。

1 つの **MR** で複数の **jar** リソースを呼び出す場合、クラスパスは`-classpath ./xxxx1.jar,./xxxx2.jar` のように記述します。**2** つのパスは、コンマで区切ります。

4. ノードスケジューリングの設定

ノードタスク編集エリアの右側で **[Schedule]** をクリックし、**[node scheduling configuration]** ページへ移動します。 詳細は、「スケジューリングの設定」 をご参照ください。

5. ノードを送信します。

設定が完了した後、ページの左上隅にある **[Save]** をクリックするか、または **[Ctrl]** + **[S]** を押して ノードを開発環境へ送信 (およびロックの解除) します。

6. ノードタスクを発行します。

   操作の詳細については、「リリースの管理」をご参照ください。

7. 本番環境でテストします。

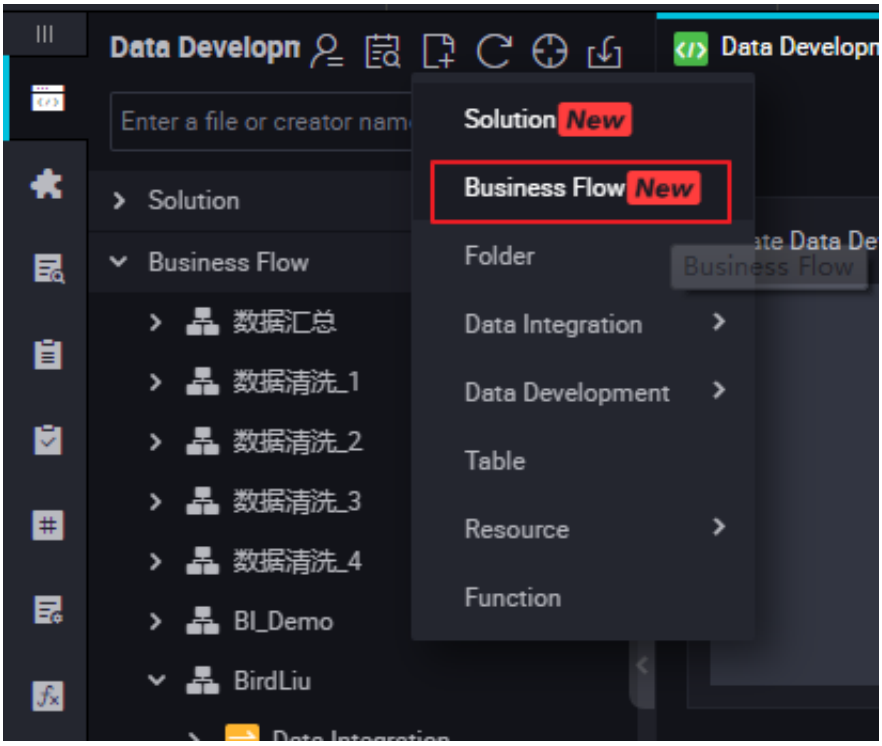
   操作の詳細については、「*#unique_20*」をご参照ください。

1.4.7 SHELL ノード

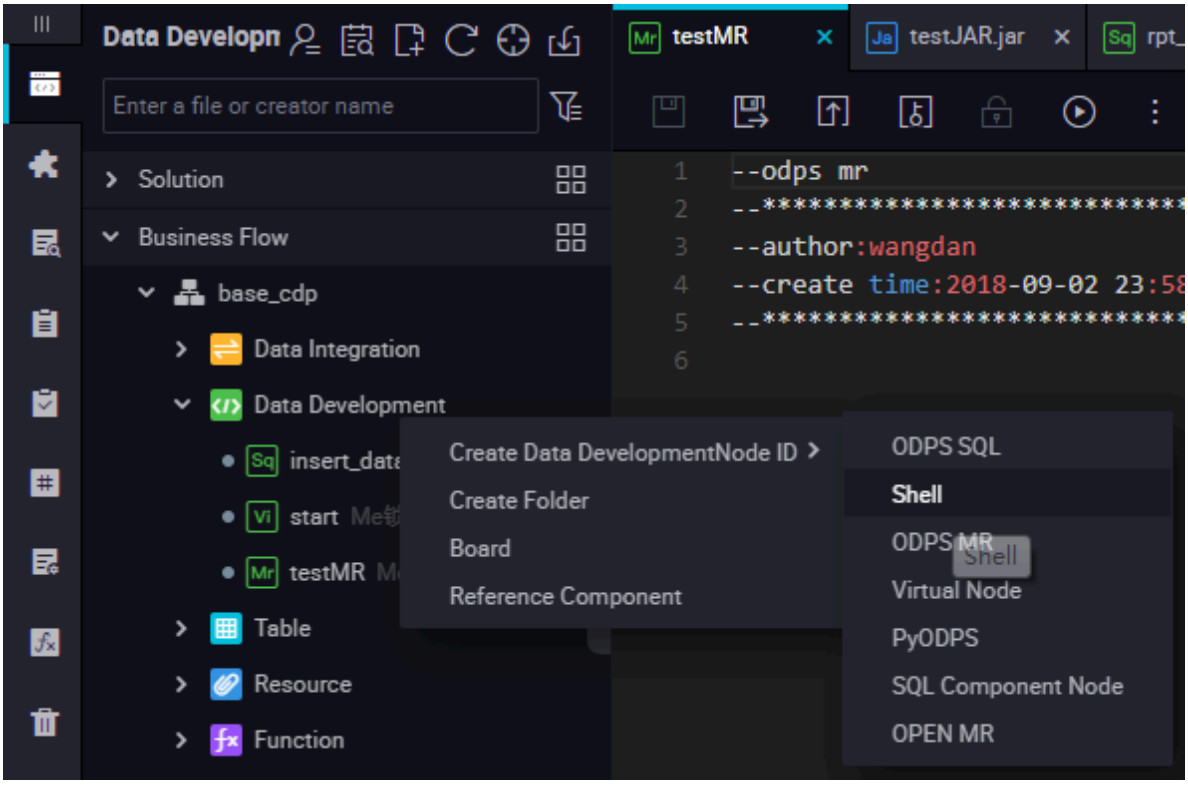
SHELL タスクは、標準の **SHELL** 構文をサポートしていますが、対話型の構文はサポートしていません。 **SHELL** タスクは、デフォルトのリソースグループで実行することができます。 **IP** アドレスやドメイン名へアクセスする場合、**[Project Configuration]** を選択し、**IP** アドレスまたはドメイン名をホワイトリストに追加します。

手順

1. **[Data Development]** の **[Business Flow]** を右クリックし、**[Create Business Flow]** を選択します。

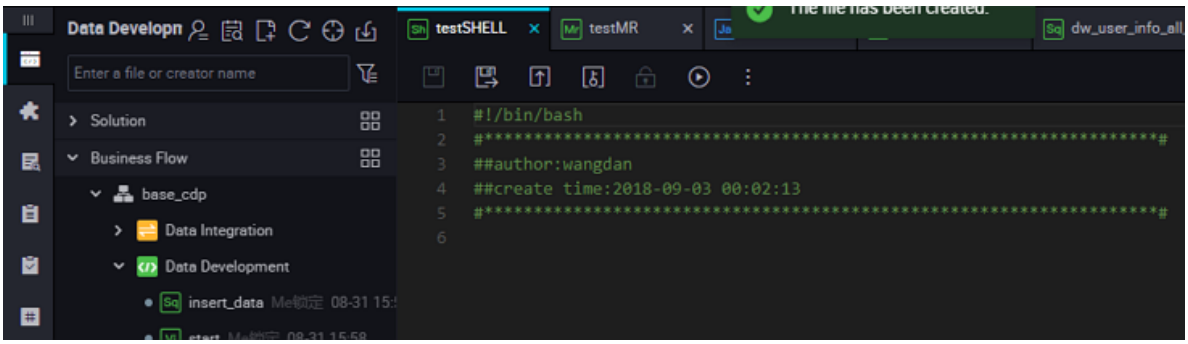


2. **[Data Development]** を右クリックし、**[Create Data Development Node]** > **[SHELL]** の順に選択します。



3. ノードタイプを「**SHELL**」に設定し、ノード名を入力します。ターゲットフォルダを選択し、**[Submit]** をクリックします。
4. ノードコードを編集します。

   SHELL ノードコード編集ページへ移動し、コードを編集します。



   SHELL 文でシステムスケジューリングパラメーターを呼び出す場合、**SHELL** 文を次のとおりコンパイルします。

```
echo "$1 $2 $3"
```

   注：

パラメーター 1 パラメーター 2 ... 複数のパラメーターはスペースで区切ります。 システムのスケジューリングパラメーターの使用方法に関する詳細は、「パラメーターの設定」をご参照ください。

5. ノードスケジューリングの設定

   ノードタスク編集エリアの右側で **[Scheduling Configuration]** をクリックし、**[node scheduling configuration]** ページへ移動します。 詳細は、「スケジューリングの設定」をご参照ください。

6. ノードを送信します。

   設定が完了した後、ページの左上隅にある **[Save]** をクリックするか、または **[Ctrl]** + **[S]** を押してノードを開発環境へ送信 (およびロックの解除) します。

7. ノードタスクをリリースします。

   操作の詳細については、「リリースの管理」をご参照ください。

8. 本番環境でテストします。

   操作の詳細については、「*#unique_20*」をご参照ください。

使用例

SHELL を使ってデータベースへ接続します。

- **Alibaba Cloud** でデータベースを作成し、リージョンが中国 (上海) の場合、データベースへ接続するには、次のホワイトリスト内の **IP** アドレスでデータベースを開きます。

  10.152.69.0/24、10.153.136.0/24、10.143.32.0/24、120.27.160.26、10.46.67.156、120.27.160.81,10.46.64.81、121.43.110.160、10.117.39.238、121.43.112.137、10.117.28.203、118.178.84.74、10.27.63.41、118.178.56.228,10.27.63.60、118.178.59.233、10.27.63.38、118.178.142.154、10.27.63.15、100.64.0.0/8

  注:
  Alibaba Cloud でデータベースを作成したが、リージョンが中国 (上海) ではない場合、インターネットを使用するか、またはデータベースと同じリージョンで **ECS** インスタンスをスケジューリングリソースとして購入し、カスタムリソースグループで **SHELL** タスクを実行することを推奨します。

- データベースをローカルで作成した場合は、インターネットを使って上記のホワイトリスト内の **IP** アドレスでデータベースを開くことを推奨します。

  注:

カスタムリソースグループを使って SHELL タスクを実行する場合、カスタムリソースグループマシンの IP アドレスを上記のホワイトリストへ追加する必要があります。

1.4.8 PyODPS ノード

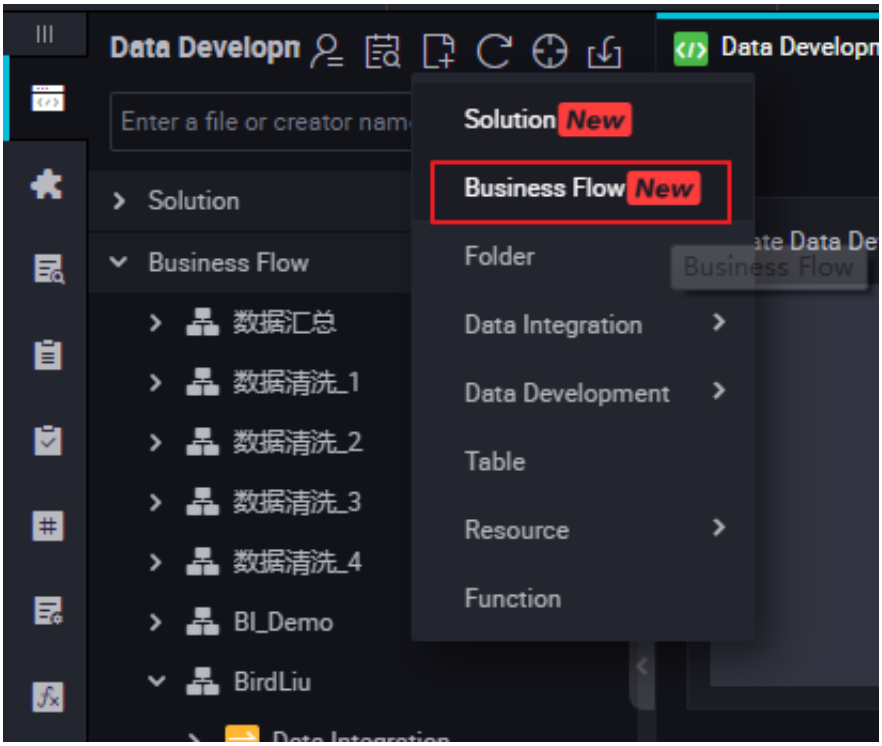
DataWorks では、PyODPS タスクタイプを提供し MaxCompute の Python SDK を統合することもできます。 Python コードを直接編集し、DataWorks の PyODPS ノードで MaxCompute を操作することができます。

MaxCompute では、MaxCompute を操作するために使用することができる *Python SDK* が提供されます。

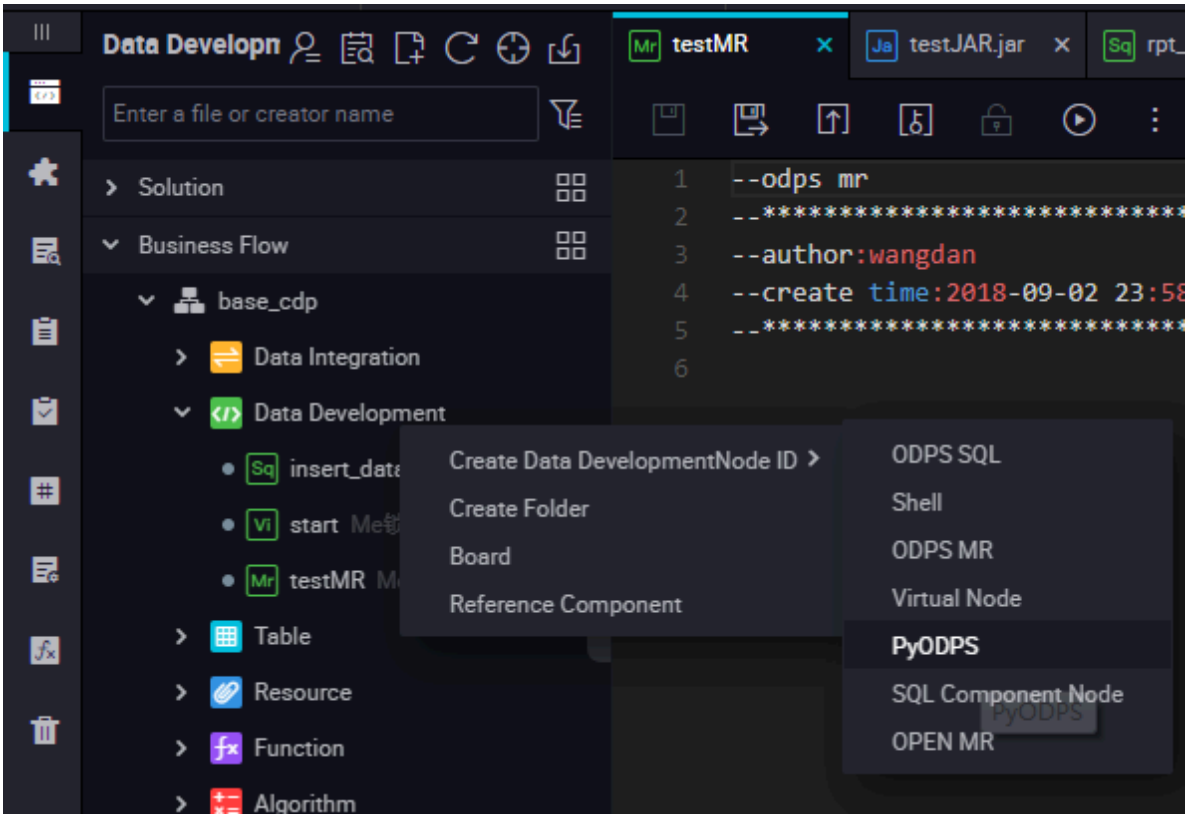
注：
Python 2.7 は基礎となるレイヤーで使用されます。 PyODPS ノードプロセスのデータサイズは、50MB を超えることはできません。使用するメモリ量は 1GB を超えることができません。

PyODPS ノードの作成

1. [Data Development] で [Business Flow] を右クリックし、[Create Business Flow] を選択します。



2. **[Data Development]** を右クリックし、**[Create Data Development Node]** > **[PyODPS]** を選択します。



3. **PyODPS** ノードを編集します。

   a. **ODPS** ポータル

   DataWorks では、**PyODPS** ノードはグローバル変数 **ODPS** または **ODPS** エントリである **o** が含まれています。手動で **ODPS** エントリを定義する必要がありません。

   ```
   print(odps.exist_table('PyODPS_iris'))
   ```

   b. **SQL** 文を実行します。

   PyODPS では、**ODPS SQL** クエリをサポートし、実行結果を読むことができます。**execute_sql** または **run_sql** メソッドのリターン値は実行中インスタンスです。

   > 注：
   > **ODPS** コンソールで実行できるコマンドのすべてが **ODPS** で受け入れられる **SQL** 文ではありません。 非 **DDL/DML** 文を呼び出すには、他の方法を使用する必要があります。 たとえば、**run_security_query** メソッドを使って **GRANT** 文または **REVOKE** 文を呼び

出します。**run_xflow** または **execute_xflow** メソッドを使って **PAI** コマンドを呼び出します。

```
o.execute_sql('select * from dual') # Run the SQL statements in
synchronous mode. Blocking continues until execution of the SQL
statement is completed.
instance = o.runsql('select * from dual') # Run the SQL
statements in asynchronous mode.
print(instance.getlogview_address()) # Obtain the logview address
.
instance.waitforsuccess() # Blocking continues until execution of
 the SQL statement is completed.
```

c. ランタイムパラメーターを設定します。

ランタイムパラメーターは必ず設定する必要があります。 パラメータータイプ **dict** を使ってヒントパラメーターを設定することもできます。

```
o.execute_sql('select * from PyODPS_iris', hints={'odps.sql.mapper
.split.size': 16})
```

sql. 設定をグローバル設定へ追加した後、関連のランタイムパラメーターがそれぞれの実行中の **python** へ追加されます。

```
from odps import options
options.sql.settings = {'odps.sql.mapper.split.size': 16}
o.execute_sql('select * from PyODPS_iris')  # "hints" is added
based on the global configuration.
```

d. **SQL** 文の実行結果を読み取ります。

SQL 文を実行するインスタンスは、**open_reader** 操作を直接行うことができます。 この場合、構造化されたデータは **SQL** 文の実行結果として返されます。

```
with o.execute_sql('select * from dual').open_reader() as reader:
for record in reader:  # Process each record.
```

別の場合では、**desc** が **SQL** 文で実行される場合があります。 この場合、オリジナルの **SQL** 文の実行結果は、**reader.raw** 属性から取得されます。

```
with o.execute_sql('desc dual').open_reader() as reader:
print(reader.raw)
```

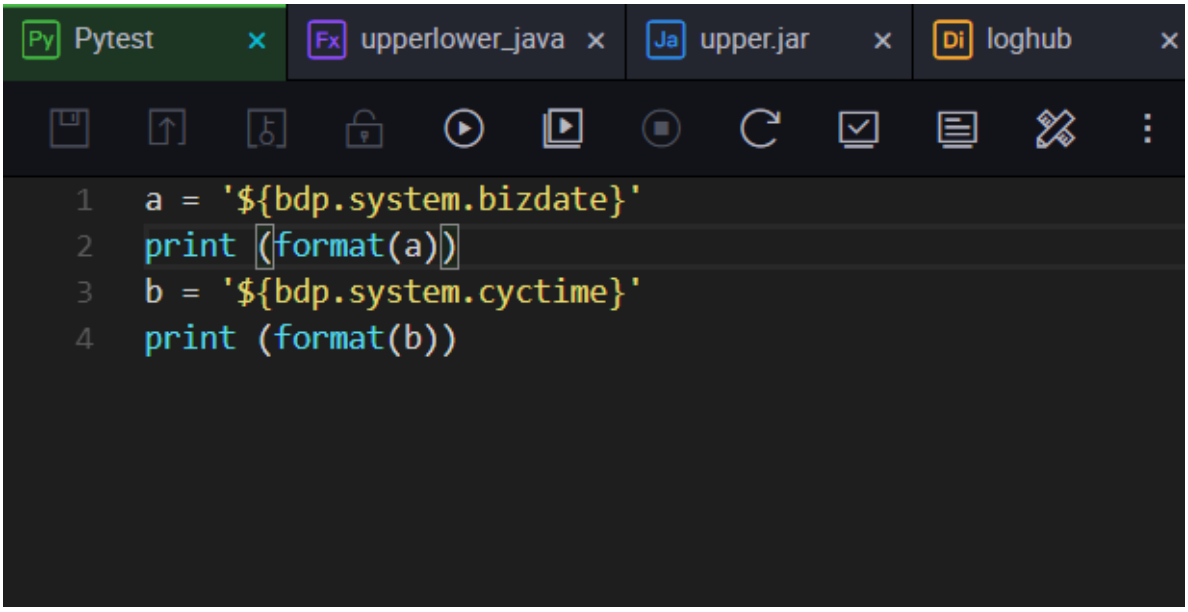
 注：

ユーザーが定義したスケジューリングパラメーターは、データ開発に使用されます。

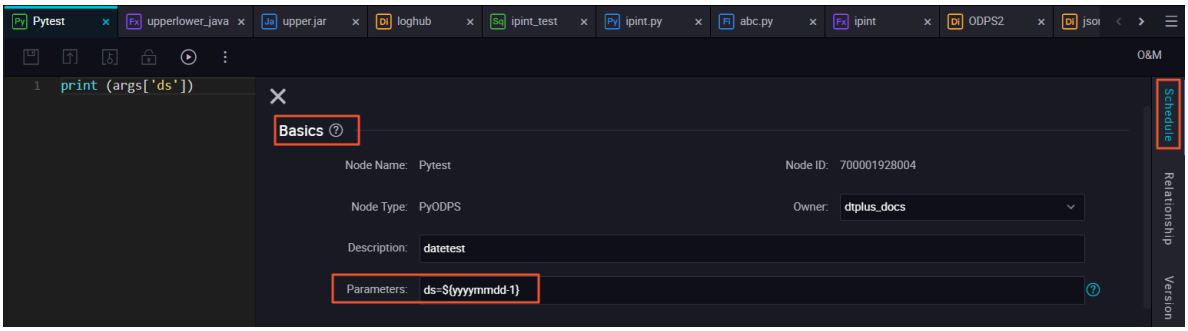
PyODPS ノードがページ上で直接トリガーされる場合、時間は明確に指定する必要があり

ます。 **PyODPS** ノードの時間は、**SQL** ノードの時間のように直接置き換えることができません。

次のようにシステムパラメーターを設定することができます。



次のようにユーザー定義パラメーターを設定することができます。



4. ノードスケジューリングの設定

   ノードタスク編集エリアの右側で **[Schedule]** をクリックし、**[node scheduling configuration]** ページへ移動します。 詳細は、「スケジューリングの設定」 をご参照ください。

5. ノードを送信します。

   設定が完了した後、ページの左上隅にある **[Save]** をクリックする、または **ctrl + S** を押してノードを開発環境へ送信 (およびロックの解除) します。

6. ノードタスクを発行します。

   操作の詳細については、「リリースの管理」をご参照ください。

7. 運用環境でテストします。

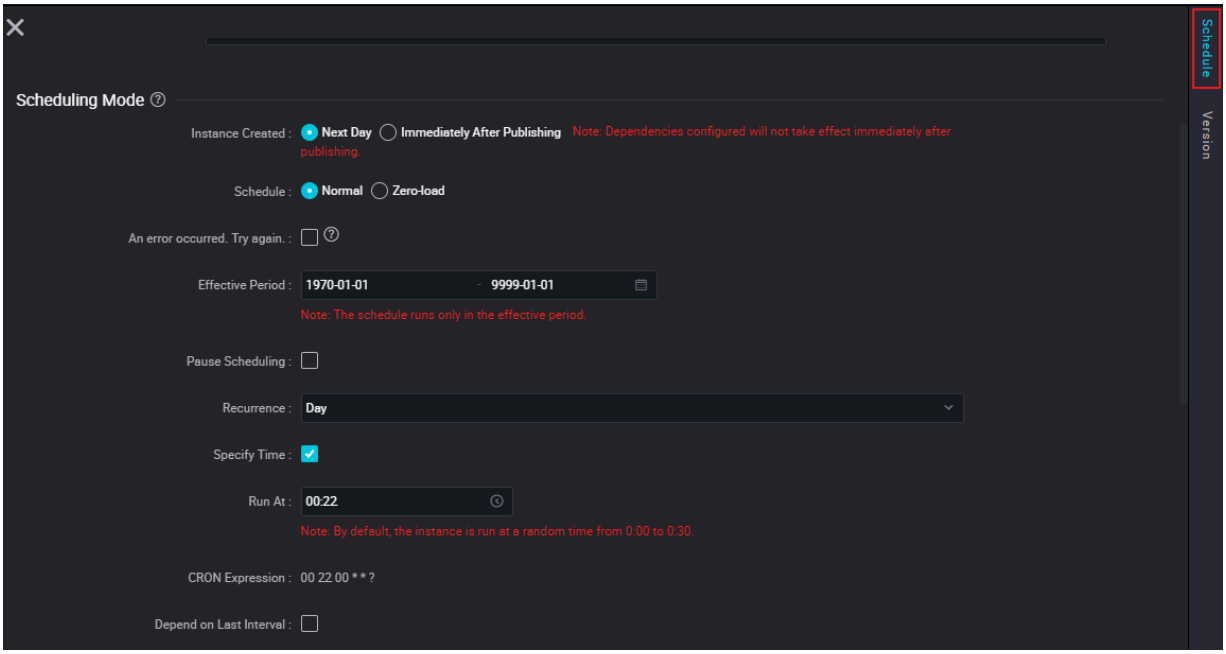
   操作の詳細については、「*#unique_20*」をご参照ください。

1.4.9 クロステナントノード

このトピックでは、異なるテナントのノードを関連付けるために通常使用されるクロステナントノードについて説明します。 クロステナントノードは送信側ノードと受信側ノードに分けられます。

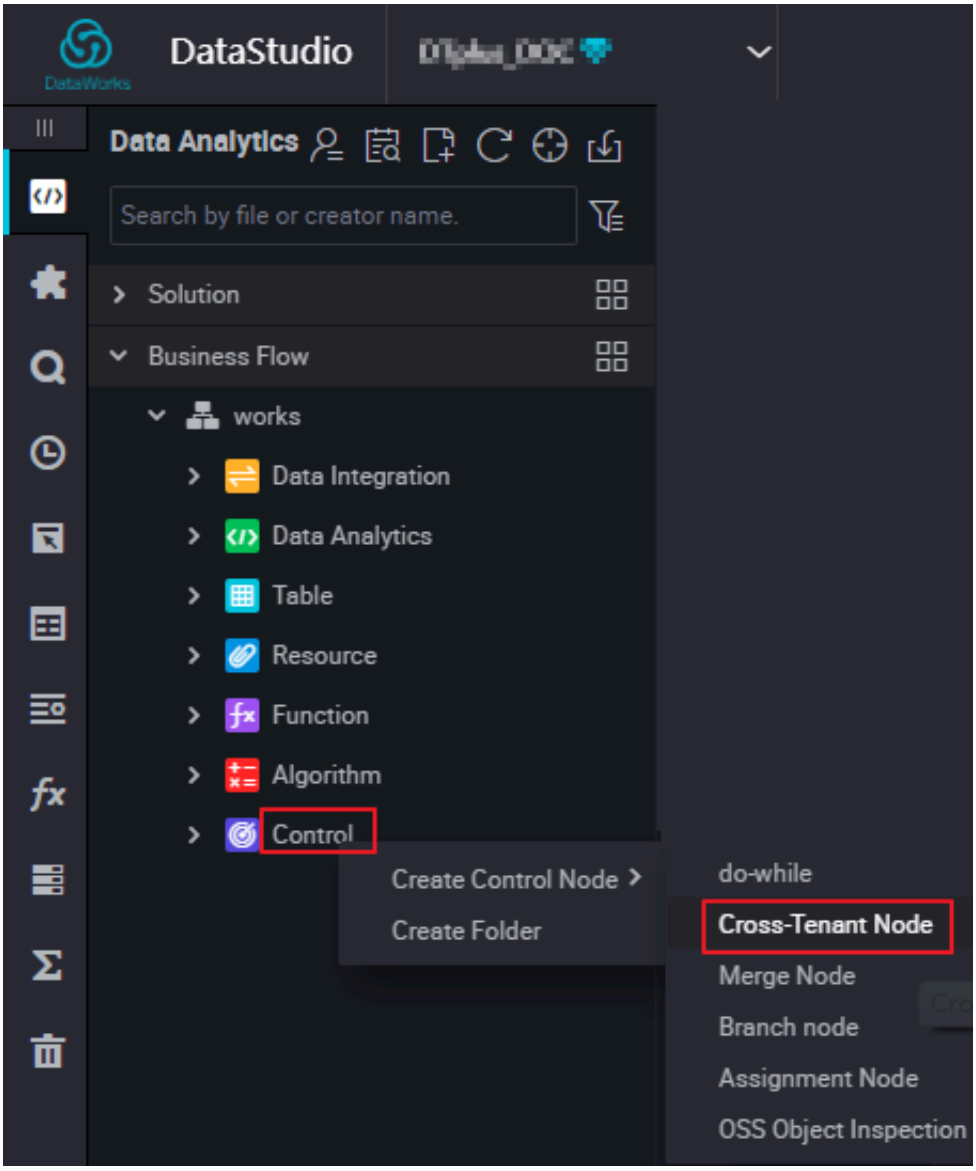
前提条件

送信側ノードと受信側ノードは同じ **Cron** 式を使用する必要があります。 **[Schedule]** > **[Scheduling Mode]**を選択して、次の図に示すように、**Cron** 式を表示します。



ノードの作成

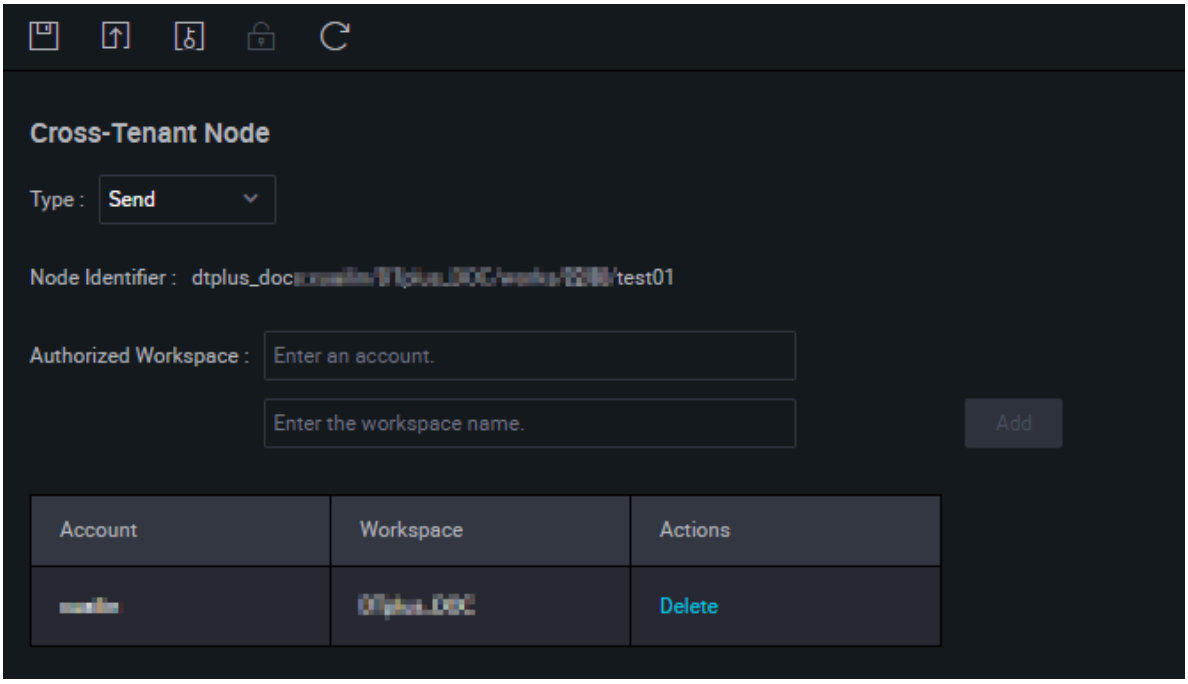
1. **[Data Studio]**ページで、**[Control]** を右クリックし、 **[Create Control Node]** > **[Cross-Tenant Node]** を選択します。



ダイアログボックスに名前を入力して**[Submit]**をクリックします。

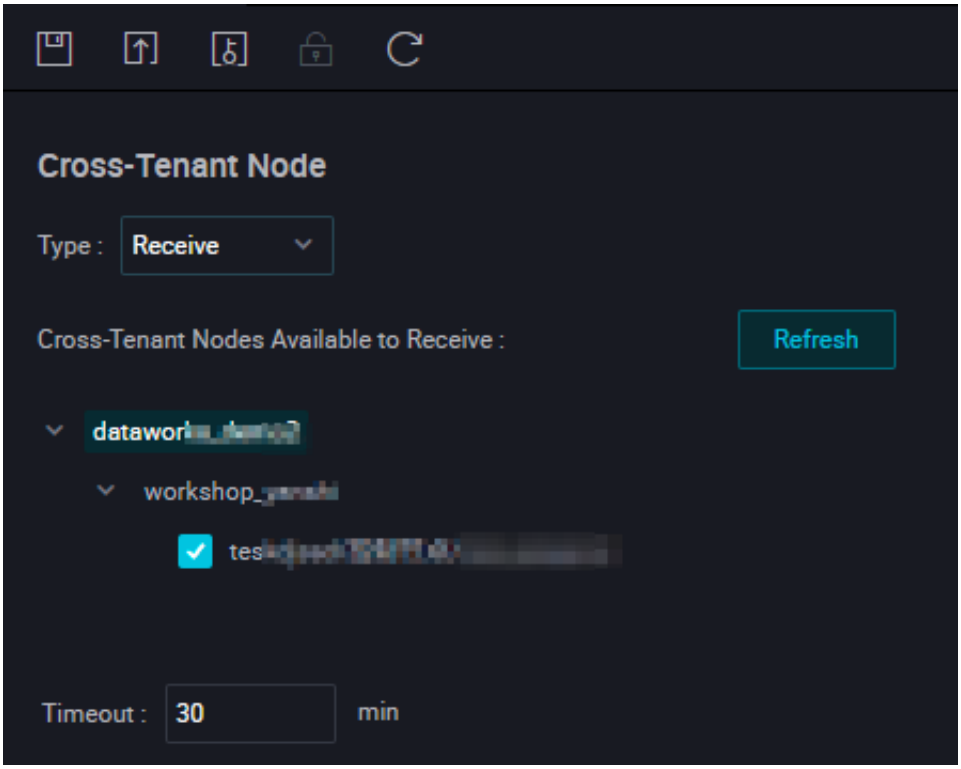
2. ノード設定を完了します。 ノードタイプを**[Send]**または**[Receive]**に設定します。 対象のワークスペースと **Alibaba Cloud** アカウントを承認します。 この例では、ノードタイプを

Send に設定します。 したがって、受信側ノードによって承認されたワークスペースとアカウントを入力する必要があります。 ノード設定が完了したら、ノードを保存して送信します。



同じ手順に従って、受信側のアカウントとワークスペースの下にコントロールノードを作成します。 ノードタイプを受信に設定します。 その後、利用可能な送信側ノードに関する情報が

表示されます。 タイムアウトタイマーも設定する必要があります。 受信側ノードの実行が開始されると、タイムアウトタイマーが再起動します。



送信側ノードはメッセージをメッセージセンターに送信し、メッセージが正常に配信された後でメッセージの実行を開始します。 受信側ノードは、メッセージセンターから継続的にメッセージを引き出します。 受信側ノードは、タイムアウト期間内にメッセージを正常にプルしたときに実行を開始します。

受信側ノードがタイムアウト期間内にメッセージを受信しないときは、タスクは失敗します。メッセージのタイムアウトは最大24時間に設定できます。

例：

2018年10月8日、定期的に作成されたインスタンスが正常に実行され、メッセージがメッセージセンターに送信されました。 2018年10月7日に設定された営業日で受信側ノードの遡及インスタンスを作成すると、受信者ノードが表示されます。

1.4.10 マージノード (Merge node)

本ページでは、マージノードの概念と作成方法、およびマージロジックの定義方法について説明します。 実例を使ったマージノードのスケジューリング設定や操作の詳細についても説明します。

概念

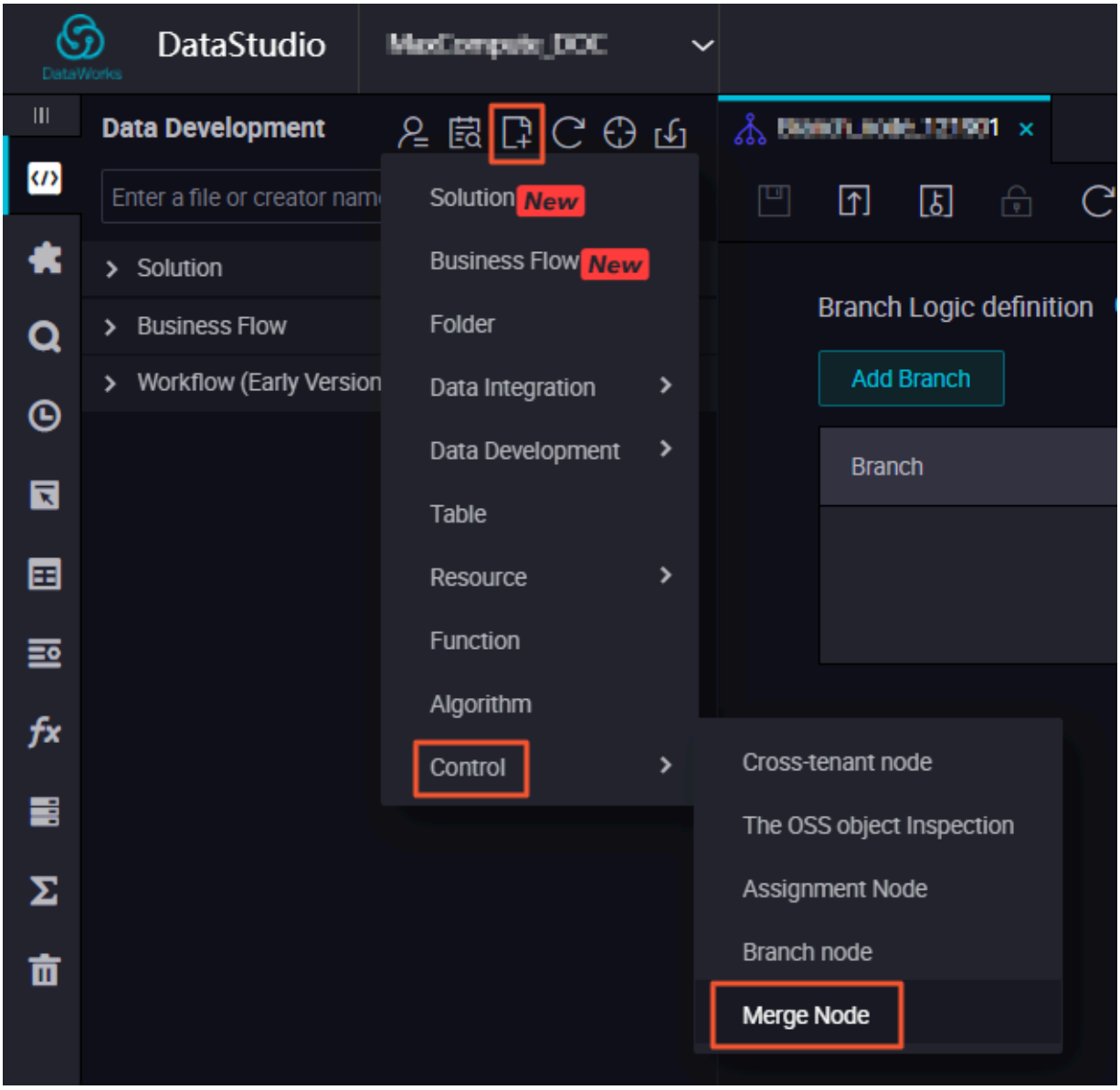
・ マージノードとは、DataStudio が提供する論理制御ファミリーノードの 1 つです。

- 依存関係のマウントや分岐ノードの下位ノードの実行トリガーに関する問題を解決するために、マージノードでは、上位ノードの実行状態を統合できます。
- 現在のマージノードの論理定義は、ノードの実行状態の選択をサポートしていませんが、分岐ノードの複数の下位ノードを正常に統合することをサポートします。下位ノードは、依存関係としてマージノードを直接マウントすることができるようになります。

たとえば、分岐ノード **C** が論理的な排他的分岐 **C1** と **C2** の **2** つを定義します。 異なる分岐では、異なるロジックを使用して、同じ**MaxCompute** テーブルへ書き込みされます。 下位ノード **B** がこの**MaxCompute** テーブルの出力に依存する場合、マージノード **J** を使って分岐を統合した後にマージノード **J** を **B** の上位の依存関係へ追加します。**B** が **C1** と **C2** に直接マウントされる場合、いつでも、**C1** と **C2** のどちらかが必ず失敗します。これは、満たされない分岐条件のためであり、スケジュールでは **B** の実行をトリガーできないためです。

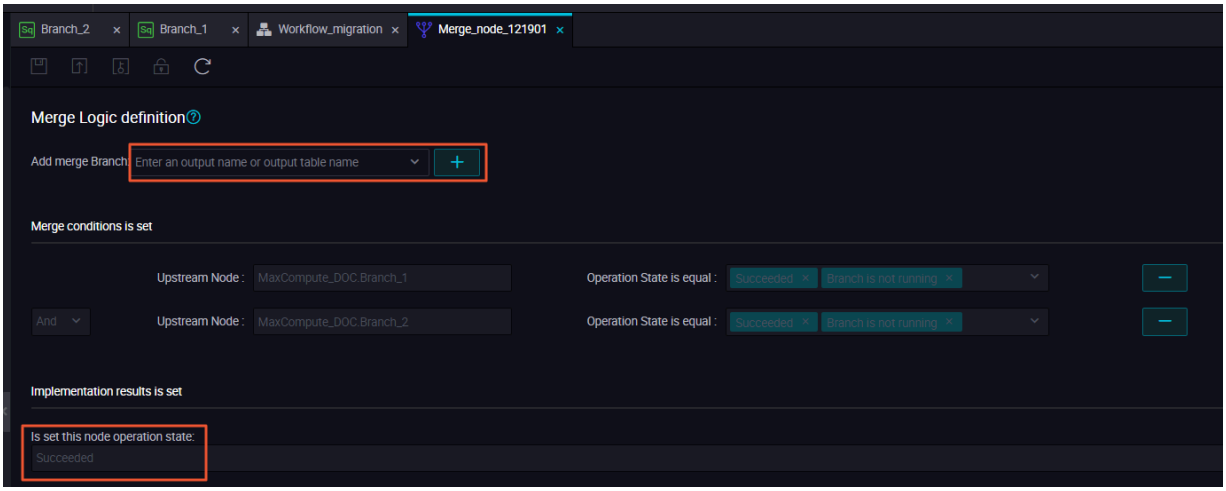
マージノードの作成

[Merge Node] は、新しいノードメニューの **[Control]** クラスディレクトリにあります。

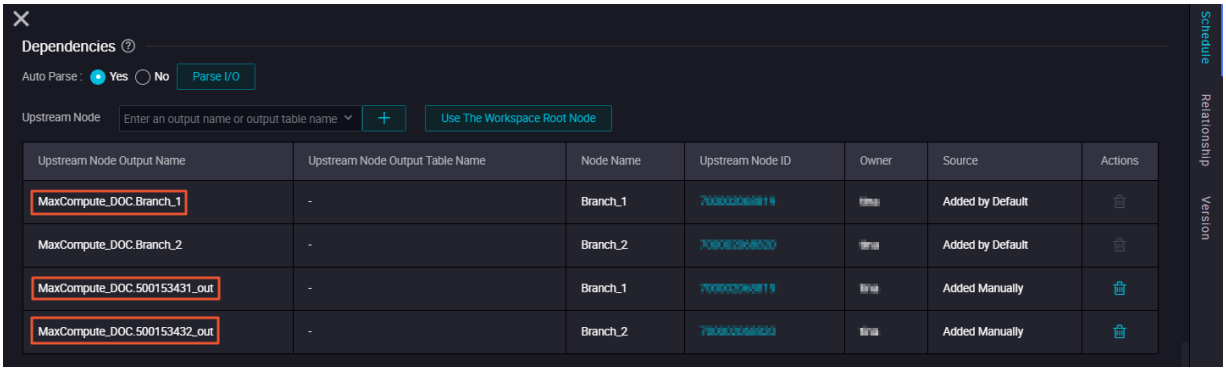


マージロジックの定義

マージ分岐を追加します。 出力名、または親ノードの出力テーブル名を入力します。**[add]** をクリックし、マージ状態のレコードを確認します。実行結果では、実行状態が表示されます。現在は、次の図に示すとおり **2** つの状態 **(Successful、Branch not running)** のみがサポートされています。

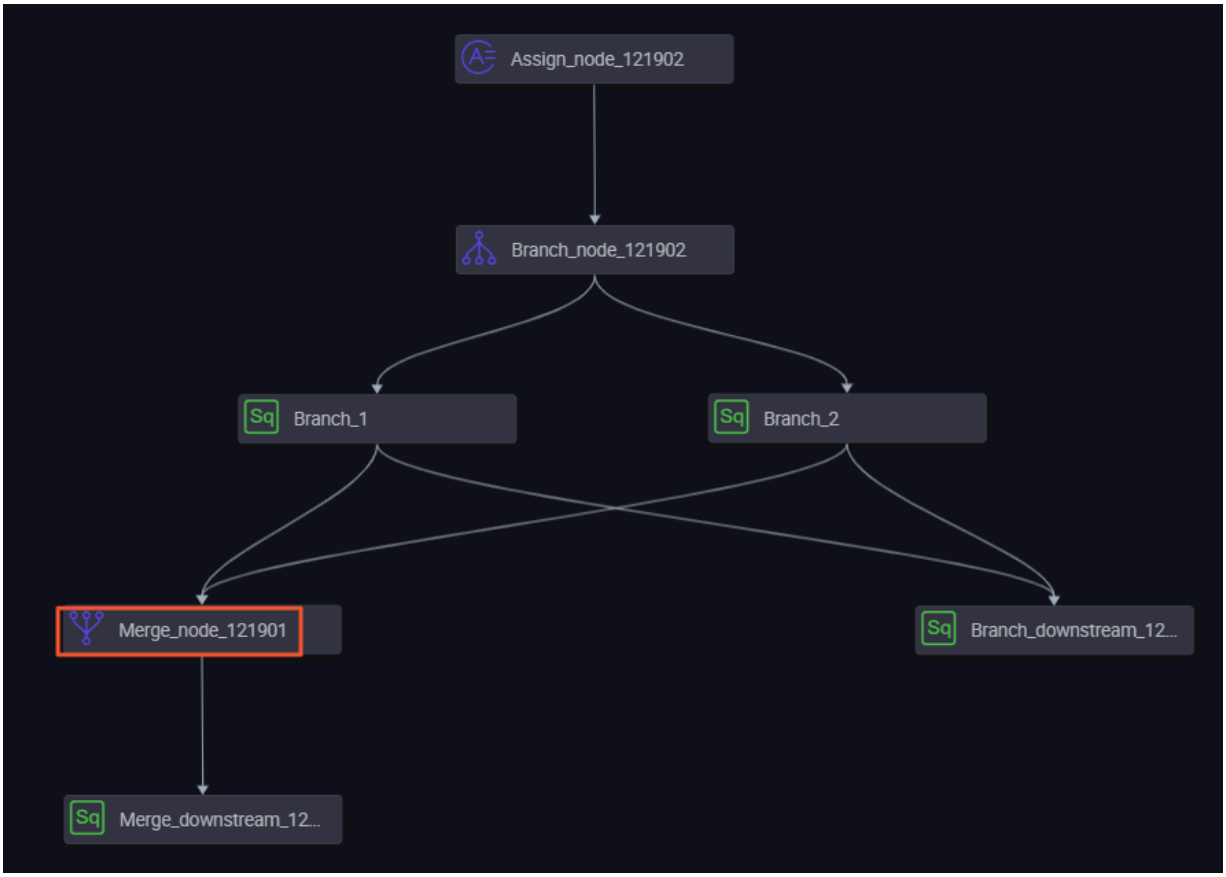


マージノードのスケジューリング属性は次の図に示すとおりです。

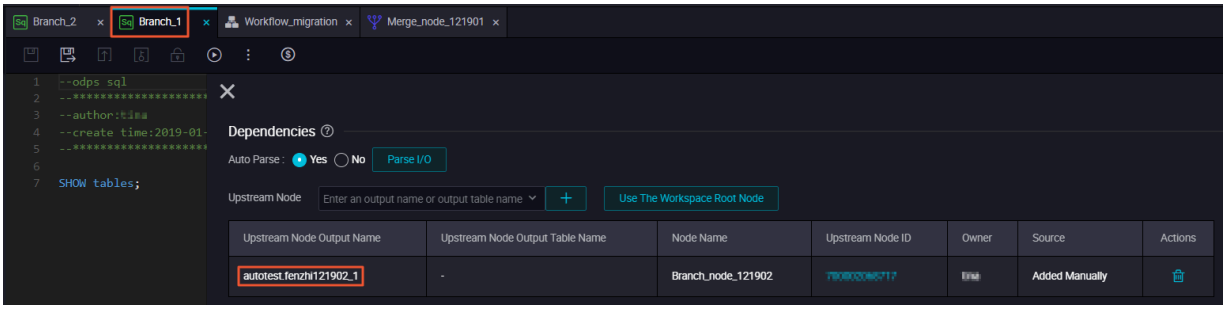


マージノードの例

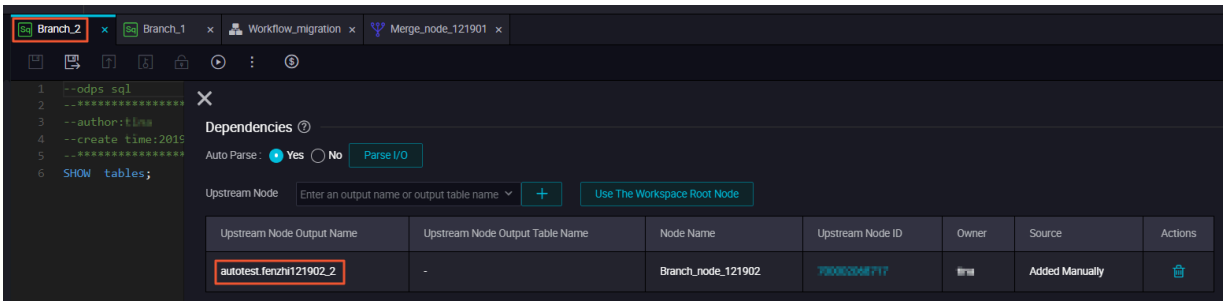
下位ノードでは、分岐ノードを上位ノードとして追加した後、対応する分岐ノード出力を選択することで、異なる条件にある分岐の方向を定義することができます。 たとえば、次の図に示すビジネスプロセスの場合、**"Branch_1"** と **"Branch_2"** はどちらも分岐ノードの下位ノードです。



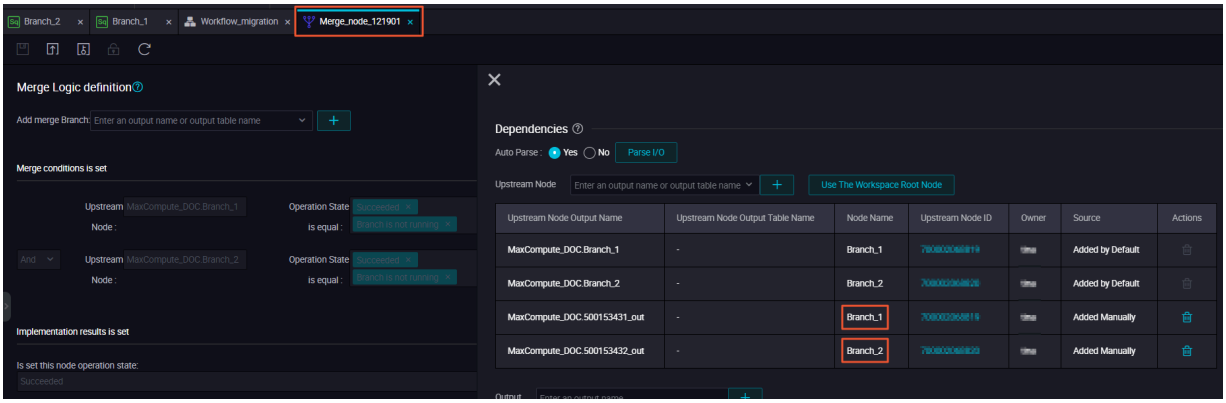
Branch_1 は、次の図に示すとおり、**autotest.fenzhi121902_1** の出力に依存します。



Branch_2 は、次の図に示すとおり、**autotest.fenzhi121902_2** の出力に依存します。



マージノードのスケジューリング属性は次の図に示すとおりです。



タスクを実行します。

分岐条件が満たされている場合、実行する分岐の下位ノードを選択します。 実行の詳細は **[Running Log]** で確認できます。

分岐条件が満たされていない場合、実行する分岐の下位ノードは選択しません。 **[Running Log]** ではノードが **"skip"** に設定されていることが確認できます。

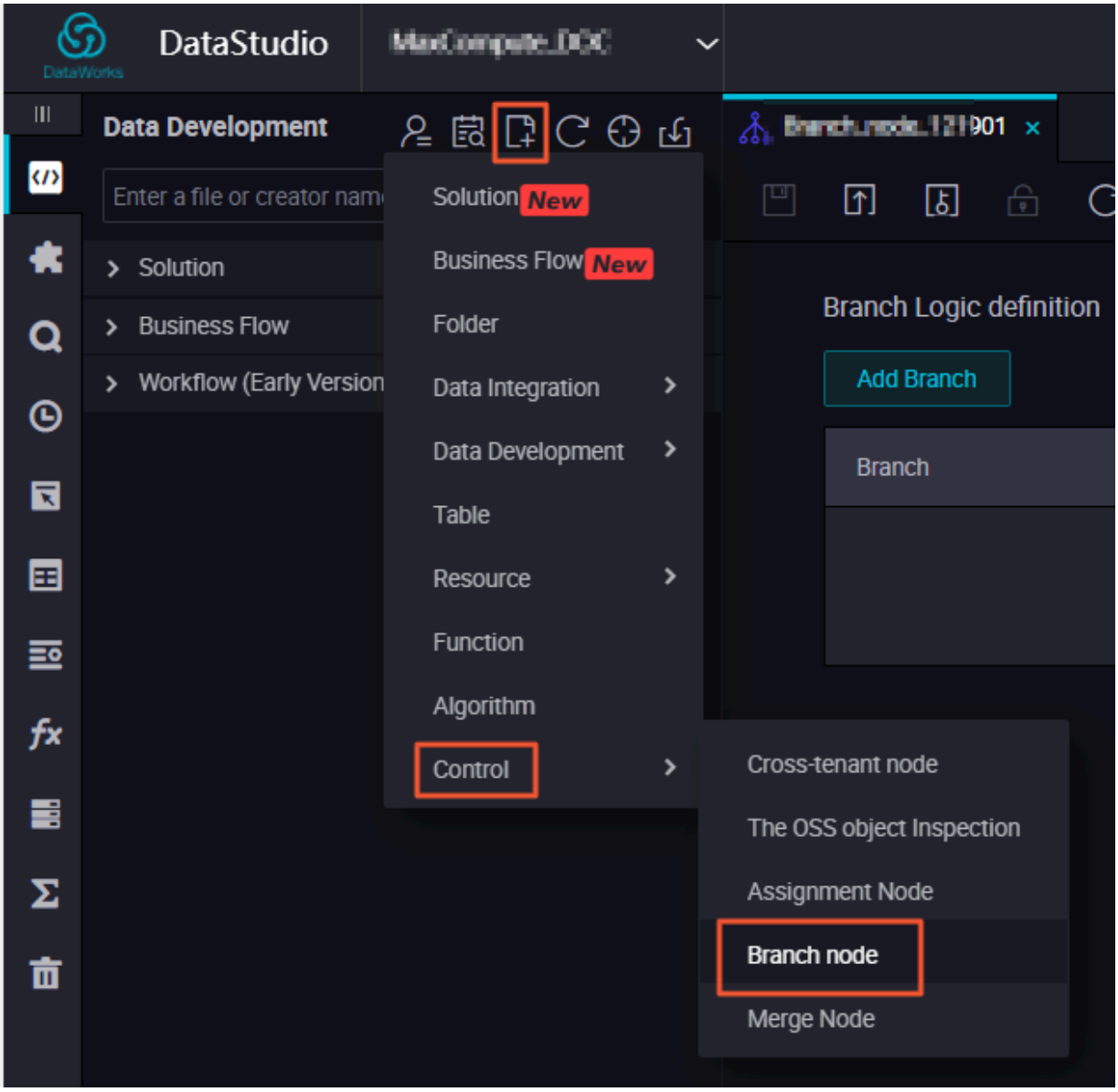
マージノードの下位ノードは正常に実行されています。

1.4.11 分岐ノード (Branch node)

分岐ノードは、**DataStudio** が提供する論理制御ファミリーノードの一つです。 分岐ノードは分岐ロジックと異なる論理条件にある下位分岐の方向を定義することができます。

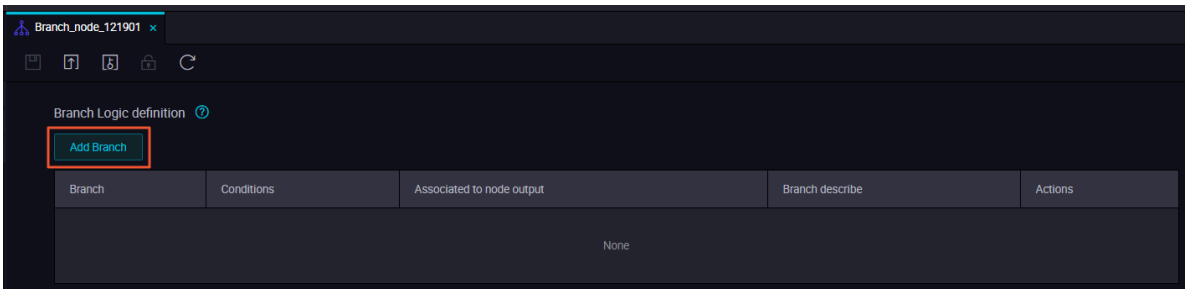
分岐ノードの作成

次の図に示すとおり、**[Branch node]** は、新しいノードメニューの **[Control]** クラスディレクトリにあります。

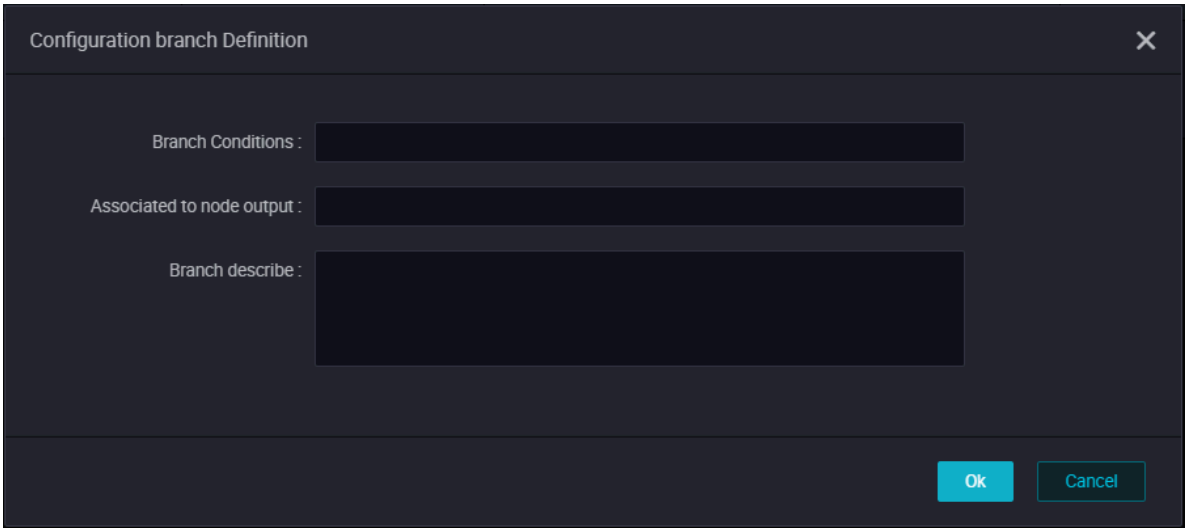


分岐ロジックの定義

1. 分岐ノードを作成した後、次の図に示すとおり **[Branch Logic definition]** ページへ移動します。



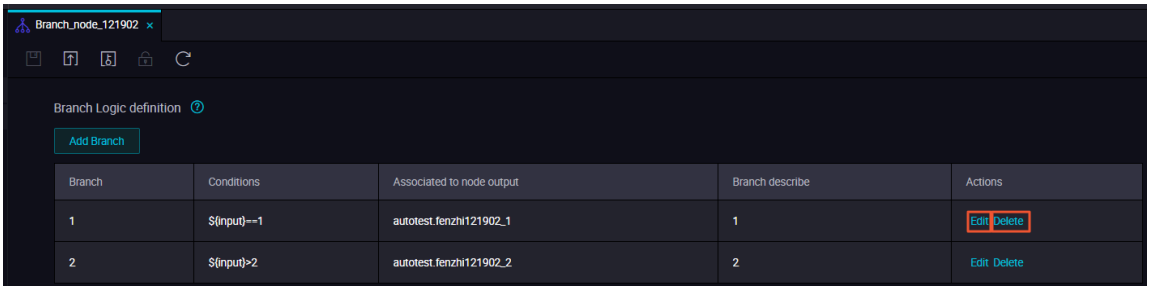
2. **[Branch Logic definition]** ページで、**[Add Branch]** ボタンを使って次の図に示すとおり **[Branch Conditions]**、**[Associated to node output]**、および**[Branch describe]** を定義します。



パラメーターは次のとおりです。

- **Branch Conditions**
  - 分岐条件は、**Python** 比較演算子に基づいて論理判断条件の定義のみをサポートします。
  - 実行状態の式の値が「**true**」の場合、対応する分岐条件は満たされていることを意味します。「**true**」でない場合は、条件が満たされていません。
  - 実行状態の式に解析エラーが報告される場合は、分岐ノード全体の実行状態が失敗するように設定されています。
  - 分岐条件は、ノードコンテキストで定義されているグローバル変数とパラメーターの使用をサポートします。図にある `${Input}` のように、分岐ノードで定義されたノード入力パラメーターとすることができます。
- **Associated to node output**
  - ノード出力は、分岐ノードの下位ノードの依存関係をマウントするために使用します。
  - 分岐条件を満たす場合、関連するノード出力にマウントされる下位ノードは、実行するために選択されます (依存関係のあるその他の上位ノードのステータスもご参照ください)。
  - 分岐条件が満たされない場合、関連するノード出力にマウントされる下位ノードは、実行するためには選択されません。分岐条件を満たしていないため、下位ノードは実行中ではないステータスになります。
- **Branch describe:** 分岐定義の説明をご参照ください。

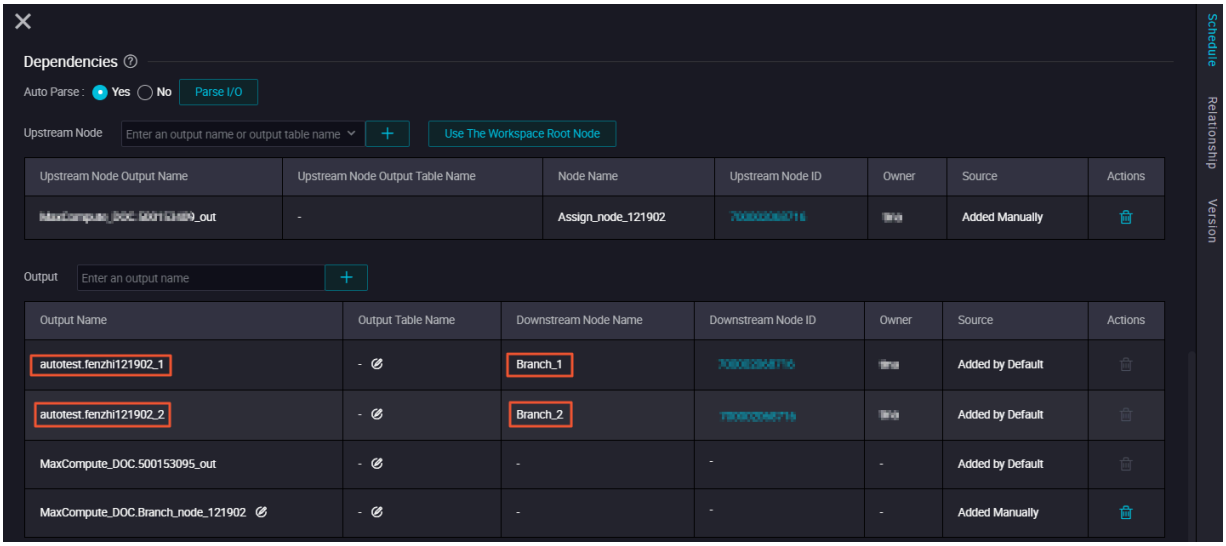
2 つの分岐の定義**:** 次の図に示すとおり、`${Input}==1` と `${Input}>2` です。



- **Edit: [Edit]** ボタンをクリックします。分岐の設定を編集したり、関連の依存関係も変更できます。
- **Delete: [Delete]** ボタンをクリックします。分岐の設定を削除したり、関連の依存関係も変更できます。

Scheduling configuration

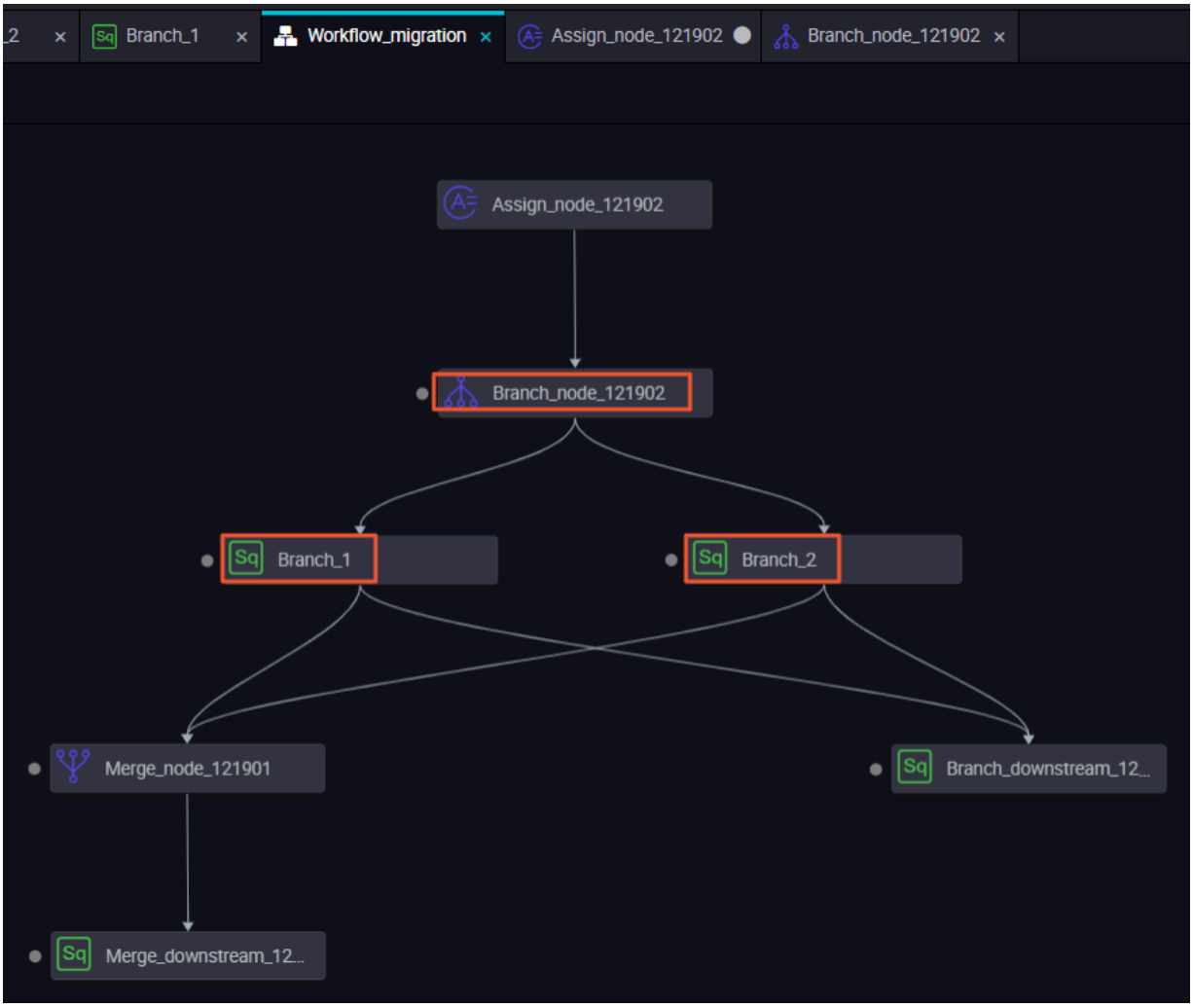
分岐条件を定義した後、**[Schedule]** のノード **[Output]** へ出力名が自動的に追加されます。下位ノードはマウントする出力名に依存します。 次の図に示すとおりです。



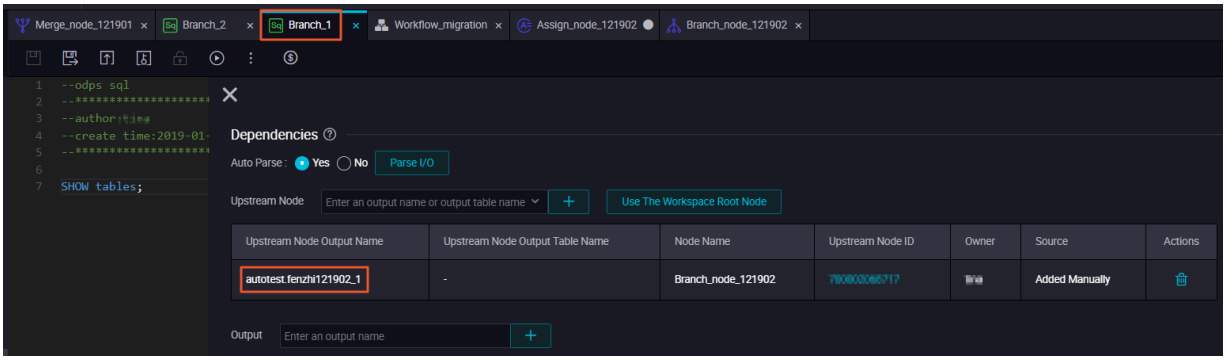
注：
書き込みで確立されたコンテキスト依存関係に対するスケジューリング設定に出力レコードがない場合は、手動で入力します。

出力ケース - 分岐ノードにマウントされる下位ノード

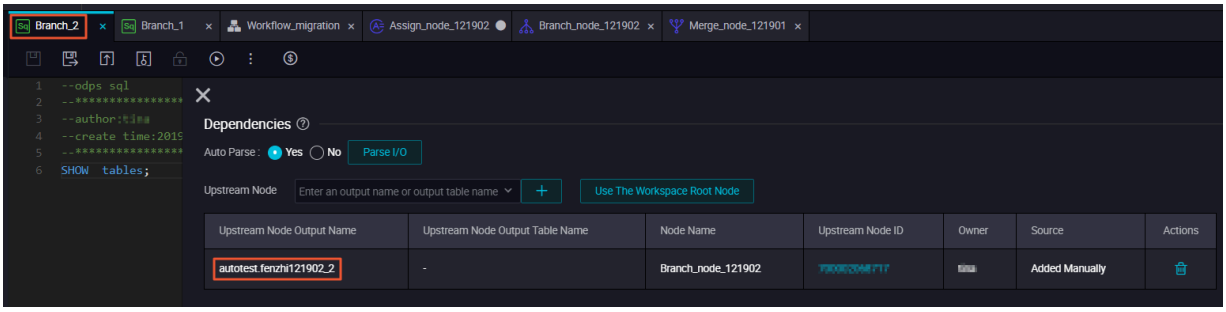
下位ノードでは、分岐ノードを上位ノードとして追加した後、対応する分岐ノードの出力を選択することで、異なる条件の分岐の方向を定義することができます。 たとえば、次の図に示すビジネスプロセスでは、**"Branch_1"** と **"Branch_2"** はどちらも分岐ノードの下位ノードです。



Branch_1 は、次の図に示すとおり、**autotest.fenzhi121902_1** の出力に依存します。



Branch_2 は、次の図に示すとおり、**autotest.fenzhi121902_2** の出力に依存します。



スケジューリング操作の送信

オペレーションセンターへ実行するディスパッチを送信します。分岐ノードが条件を満たします **(autotest.fenzhi121902_1** に依存します**)**。したがって、ログの印刷結果は次のとおりとなります。

- 分岐条件が満たされている場合、実行する分岐の下位ノードを選択します。 実行の詳細は **[Running Log]** で確認できます。
- 分岐条件が満たされていない場合、実行する分岐の下位ノードは選択しません。 **[Running Log]** ではノードが **"skip"** に設定されていることが確認できます。

追記: サポートされている Python 比較演算子

次の表では、変数 **a** は**10**、変数 **b** は **20** を前提にしています。

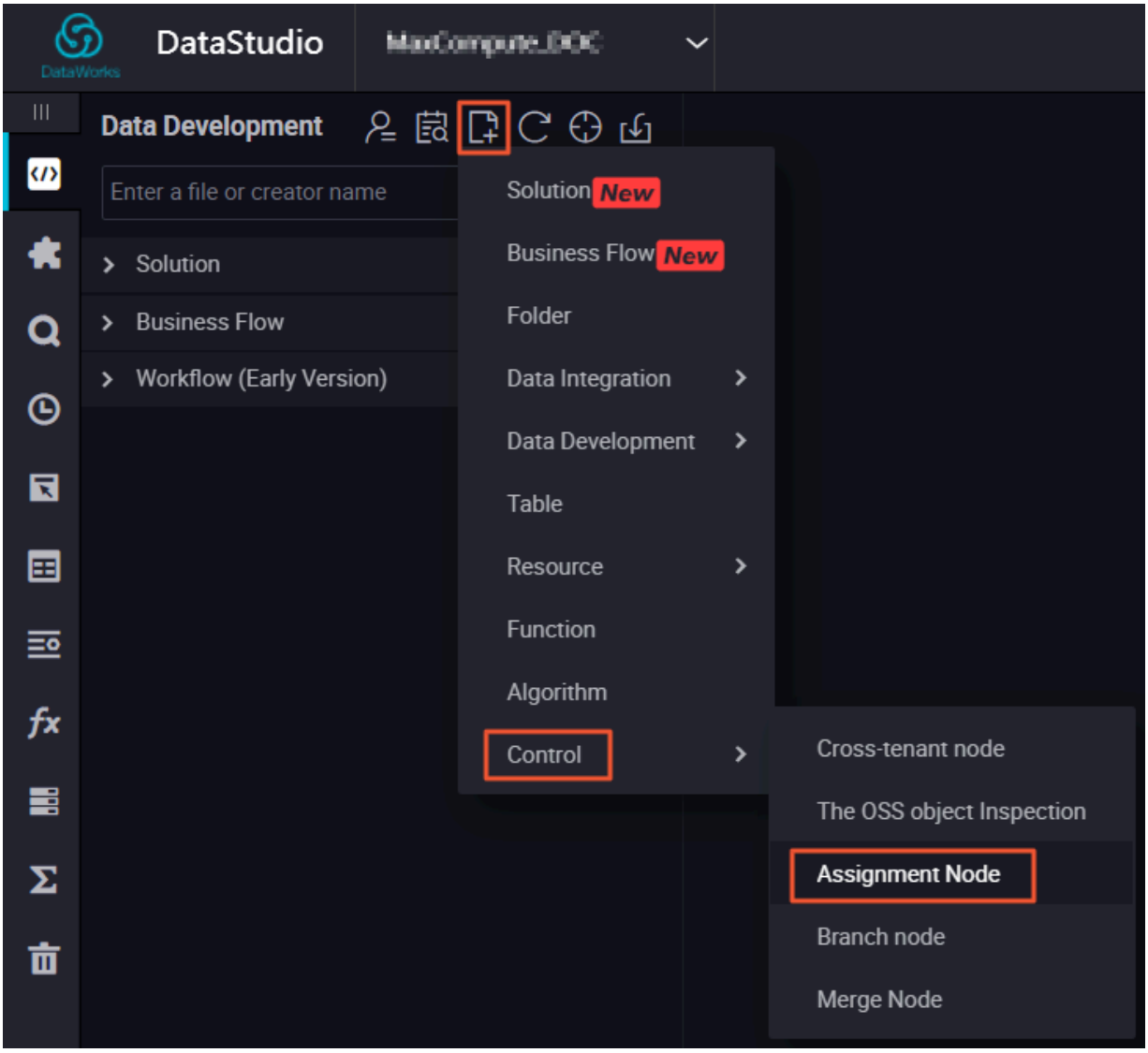
比較演算子	説明	例
==	イコール - 等値のオブジェクトを比較します。	**(a==b)** は **'false'** を返します。
!=	イコールではない - 等値ではない **2** つのオブジェクトを比較します。	**(a!=b)** は **'true'** を返します。
<>	イコールではない - 等値ではない **2** つのオブジェクトを比較します。	**(a<>b)** は **'true'** を返します。 この演算子は、**'!='** に似ています。
>	より大きい - **x** が **y** より大きいかどうかを返します。	**(a>b)** は **'false'** を返します。
<	より小さい - **x** が **y** より小さいかどうかを返します。すべての比較演算子は、**true** の場合は **1**、**false** の場合は **2** を返します。 特別な変数 **True** および **False** と同等です。	**(a<b)** は **'true'** を返します。
>=	以上 または イコール - **x** が **y** よりも大きい、または **y** と等値であるかどうかを返します。	**(a>=b)** は **'false'** を返します。
<=	以下またはイコール - **x** が **y** よりも小さい、または **y** と等値であるかどうかを返します。	**(a<=b)** は **'true'** を返します。

1.4.12 割り当てノード (Assignment node)

割り当てノード (**Assignment node**) は特別なタイプのノードです。 ノードのダウンストリームで参照しそれらの値を使用するために、ノードでコードを書き込むことによって出力パラメーターの割り当てをサポートし、ノードのコンテキストを使って組み合わせて転送します。

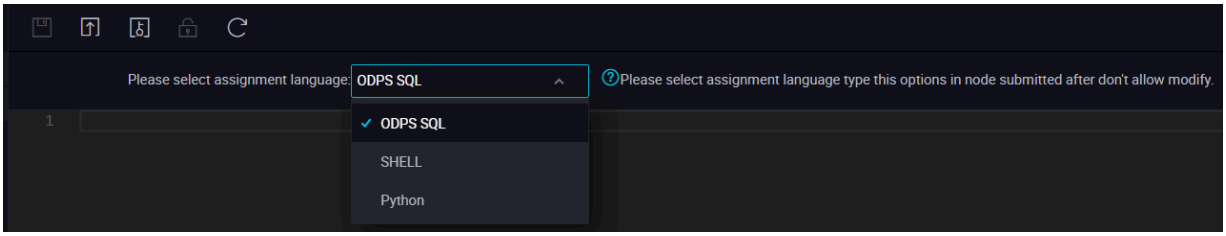
割り当てノードの作成

次の図に示すとおり、新しいノードメニューの **[Control]** クラスディレクトリには、**[Assignment Node]** が配置されています。



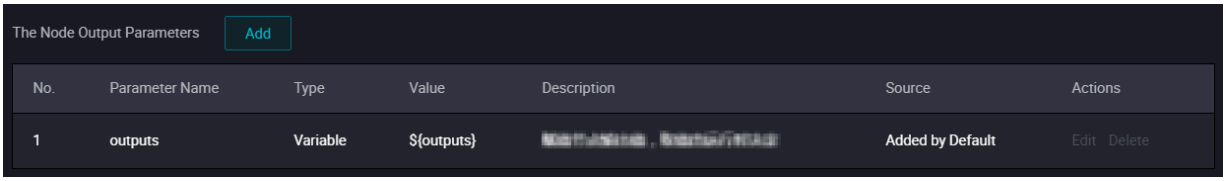
割り当てノードの値ロジックの記述

割り当てノードには、**[Node Context]** の出力に名前をつける、固定出力パラメーターがあります。 **MaxCompute**、**Shell**、**Python** の使用をサポートし、パラメーターを割り当てるコードを書き込みます。これらの値はノードコードの操作および計算結果です。 **1** つの割り当てコードに対して選択できる言語は **1** つです。



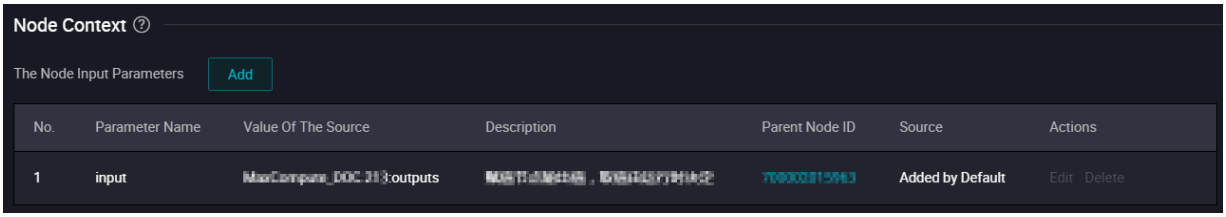
注：

- 出力パラメーターの値は、次のように、コードの最終出力行から取得されます。
  - **MaxCompute SQL** の最終行にある **SELECT** 文の出力
  - **shell** の最終行の **ECHO** 文からのデータ
  - **Python** の最終行の **PRINT** 文の出力
- 出力パラメーター値には制限があり、最大転送値は **2M** です。 割り当て文の出力がこの制限を超える場合、割り当てノードは実行に失敗します。



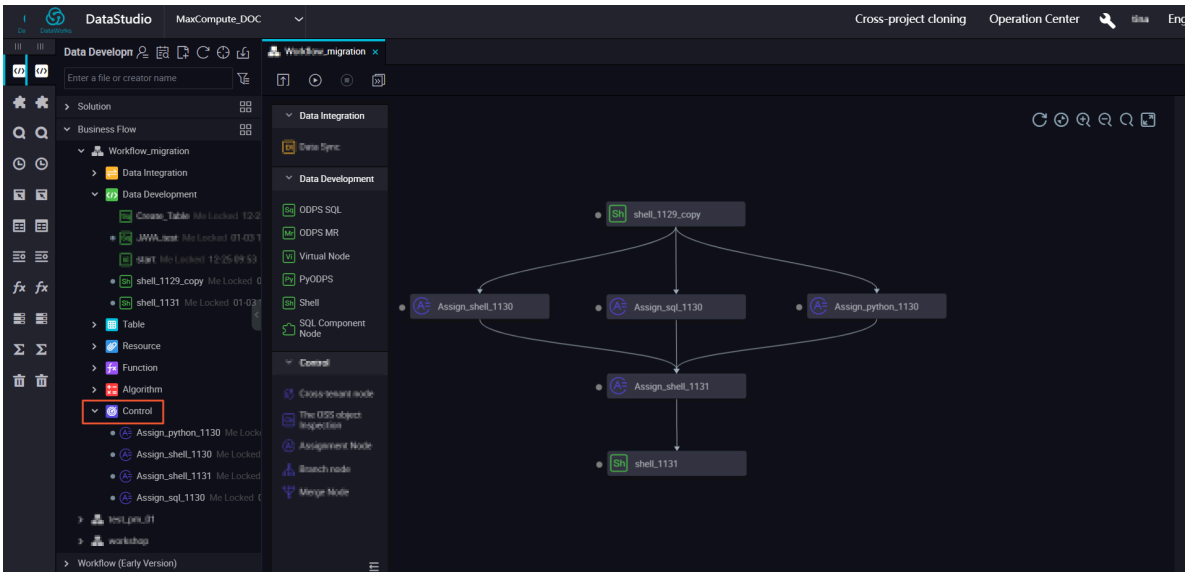
ダウンストリームノードには割り当てノードの出力を使用します。

ダウンストリームノードでは、割り当てノードをアップストリーム依存関係として追加した後、ノードコンテキストを使って割り当てノードの出力をノードの入力パラメーターとして定義します。コードでそれを参照します。アップストリーム割り当てノードの出力パラメーターの特定値が取得されます。 詳細については、「ノードのコンテキスト」をご参照ください。

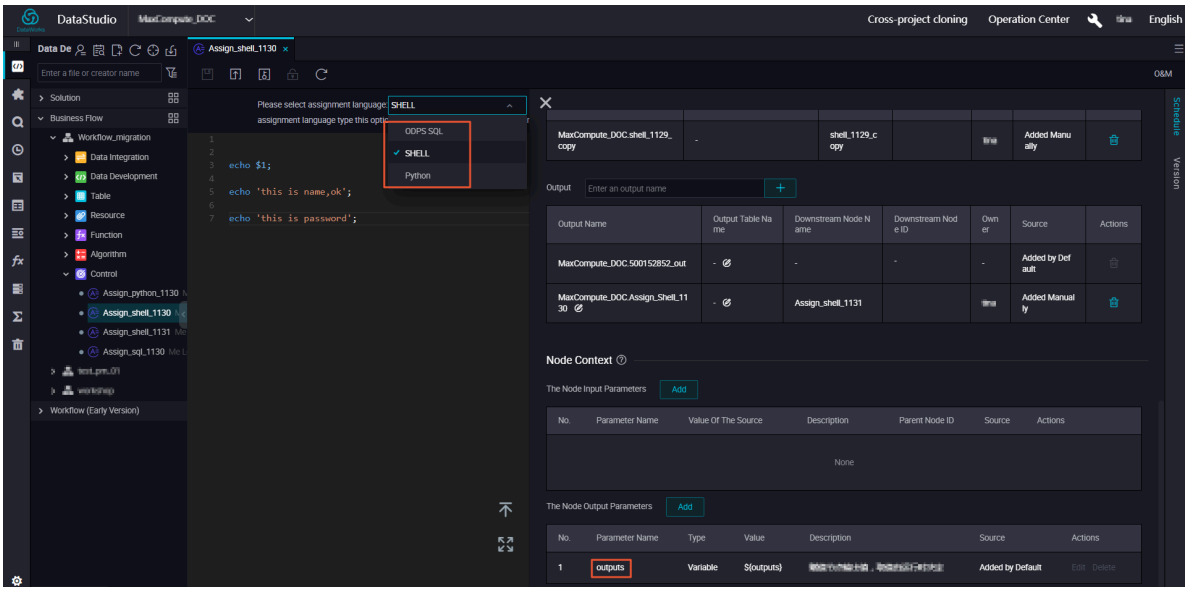


割り当てノードの例

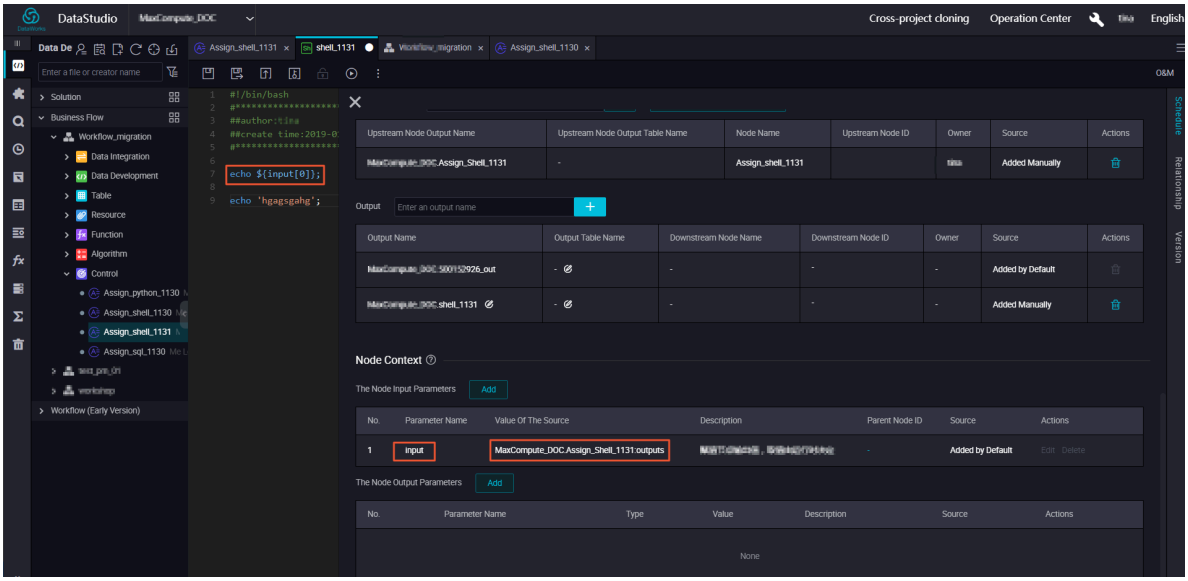
1. ビジネスプロセスを作成し、次の図に示すとおり、以下のノードをそれぞれ作成します。



2. 割り当てノードを設定する際、システムにはデフォルトで **[outputs]** パラメーターが表示されます。 実行後、関連のパラメーター結果が**[Operation Center]** > **[Properties]** > **[Context]** ページに表示されます。



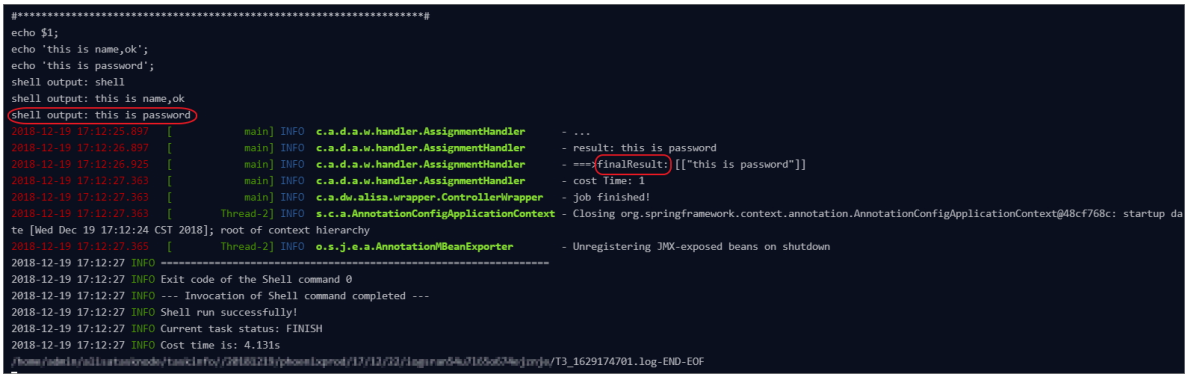
3. 次の図に示すとおり、アップストリーム **[outputs]** パラメーターをダウンストリーム入力パラメーターとして使用します。



割り当てノードタスクを実行します。

> 注：
> 通常の操作やメンテナンスでは、上記の設定パラメーターはパッチデータ操作で検証することができますが、テスト操作パラメーターは検証できません。

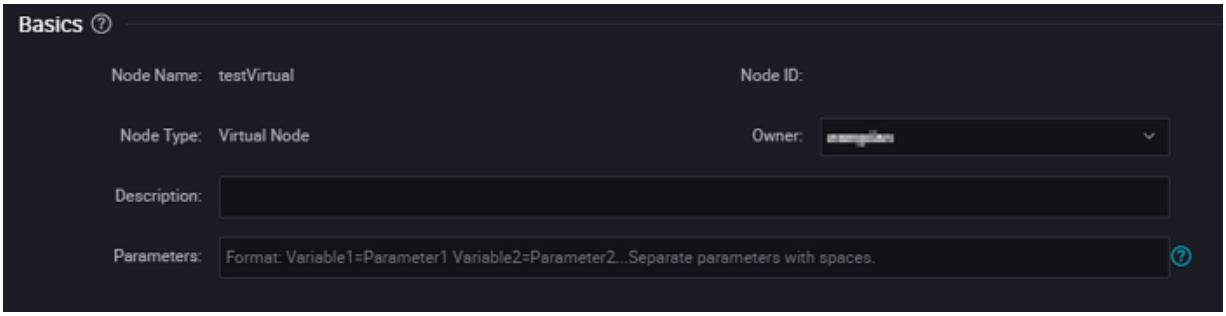
1. タスクが設定されスケジュールされると、通常、実行インスタンスが次の日に生成されます。
2. ランタイムでは、コンテキストの入出力パラメーターを表示することができます。次のリンクをクリックして入出力結果を確認します。
3. **[Running Log]** では、**"finalResult"** からのコードの最終出力を確認できます。



1.5 スケジューリングの設定

1.5.1 基本属性

下図は、基本属性を設定するためのインターフェイスです。



- **Node Name:** ワークフローノードの作成時に入力するノード名。 ノード名を編集するには、ディレクトリツリーでノード名を右クリックし、ショートカットメニューから **[Rename]** を選択します。
- **Node ID:** タスクの送信時に生成される一意のノード **ID** で、編集できません。
- **Node Type:** ワークフローノードの作成時に選択するノードタイプで、編集できません。
- **Owner:** ノードのオーナーです。 デフォルトでは、現在ログインしているユーザーが新規作成したノードのオーナーになります。 オーナーを変更するには、入力ボックスをクリックしてオーナーの名前を入力するか、または直接別のユーザーを選択します。

  > 注：
  > 別のユーザーを選択する場合、そのユーザーは現在のプロジェクトのメンバーである必要があります。

- **Description:** 通常、ビジネスとノードの目的を説明するために使用します。
- **Parameter:** タスクのスケジューリング時にコード内の変数に値を割り当てるために使用します。

  たとえば、変数 **"pt=${datetime}"** を使用してコード内で時間を指定する場合、この変数に値を割り当てることができます。 割り当てる値には、スケジューリングのビルトイン時間パラメーター **"datetime=$bizdate"** を使用できます。

さまざまなノードタイプのパラメーター値の割り当て形式

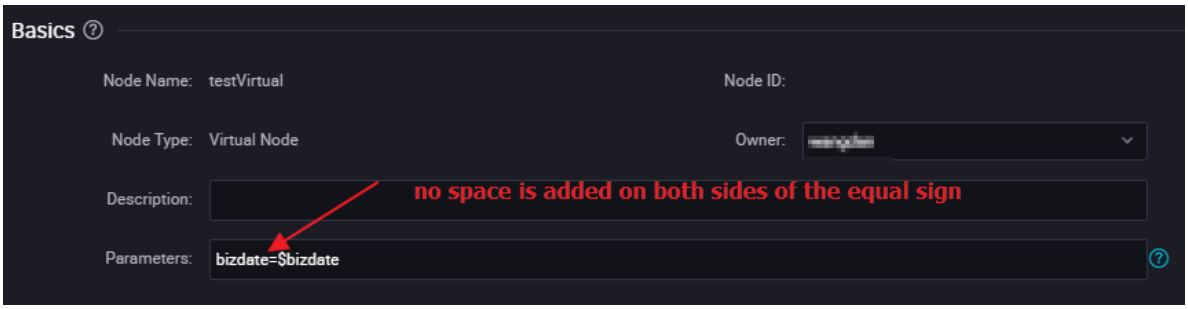
- **ODPS SQL, ODPS PL, ODPS MR** タイプ: `Variable name 1=Parameter 1 Variable name 2=Parameter 2...` 複数のパラメーターはスペースで区切ります。
- **SHELL** タイプ: `Parameter 1 Parameter 2...` 複数のパラメーターはスペースで区切ります。

頻繁に使用される時間パラメーターは、ビルトインスケジューリングパラメーターとして提供されます。 パラメーターの詳細については、「パラメーターの設定」をご参照ください。

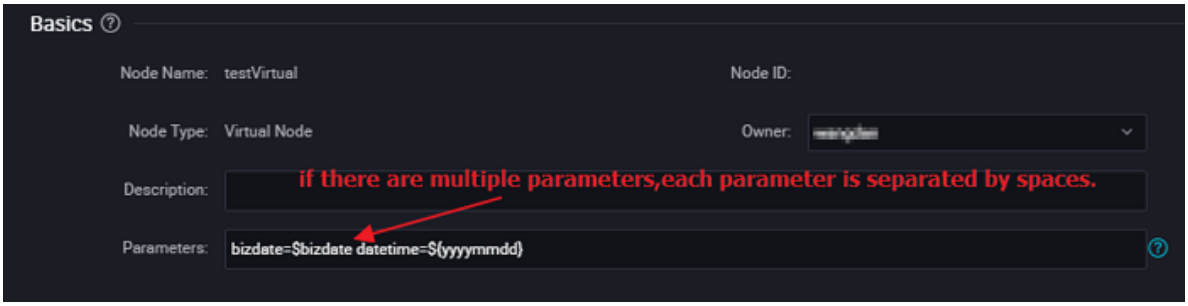
1.5.2 パラメーターの設定

スケジュールされた時間にタスクが自動的に実行される際、タスクが環境変化に動的に対応できるように、**DataWorks** にはパラメーターの設定機能が搭載されています。 パラメーターの設定を行う前に、次の **2** つの問題点について、特に注意してください。

- イコール "=" の両端にスペースを挿入することはできません。 正しい例**: bizdate=$bizdate**



- 複数のパラメーターがある場合は、スペースで区切ります。



システムパラメーター

DataWorks には **2** つのシステムパラメーターがあり、次のとおり定義します。

- **${bdp.system.cyctime}:** インスタンスのスケジュールされた実行時間を定義します。 デフォルトの形式は、**yyyymmddhh24miss** です。
- **${bdp.system.bizdate}:** インスタンスの計算が行われる営業日を定義します。 デフォルトの営業日は、実行日の一日前です。デフォルトの表示形式は、**yyyymmdd** です。

定義に基づいたランタイムの計算式と営業日の数式は次のとおりです。`Runtime = Business date + 1`

システムパラメーターを使用するには、編集ボックスでシステムパラメーターを設定せずに、コードで直接 **${bizdate}** を参照します。システムでは、コードでシステムパラメーターの参照フィールドが自動的に置換されます。

注:

定期的なタスクのスケジューリング属性は、スケジュールされたランタイムを使って設定されます。 したがって、インスタンスのスケジュールされたランタイムに基づいて営業日をバックトラックし、インスタンスのシステムパラメーター値を取得することができます。

例

毎日 **00:00** から **23:59** まで毎時間、**ODPS_SQL** タスクが実行されるように設定します。 コードでシステムパラメーターを使うには、次の文を実行します。

```
insert overwrite table tb1 partition(ds ='20180606') select
c1,c2,c3
from (
select * from tb2
where ds ='${bizdate}');
```

非 Shell ノードのスケジューリングパラメーターの設定

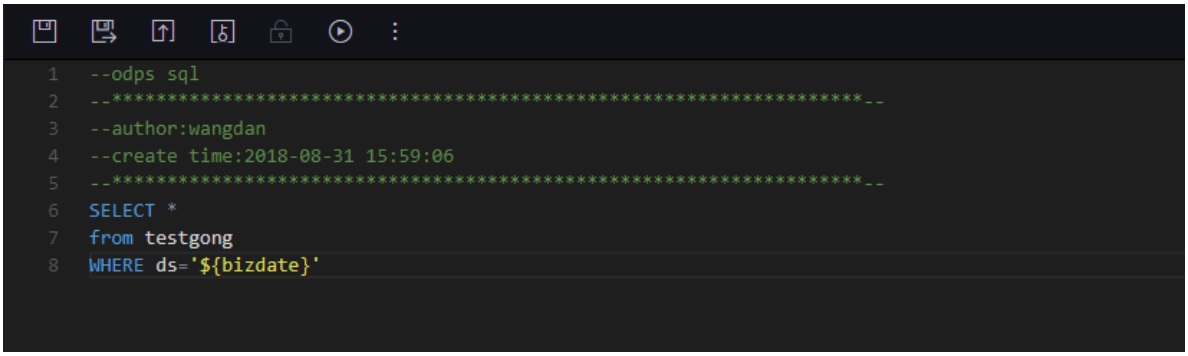
注：

SQL コードの変数名には、**a-z**、**A-Z**、数字、およびアンダースコアを使用することができます。 変数名が **"date"** の場合、値 **"$bizdate"** は自動的にこの変数へ割り当てられます。スケジューリングパラメーターの設定で値を割り当てる必要はありません。 もし別の値が割り当てられても、コードではデフォルトで自動的に値 **"$bizdate"** が割り当てられるため、この値はコードでは使用されません。

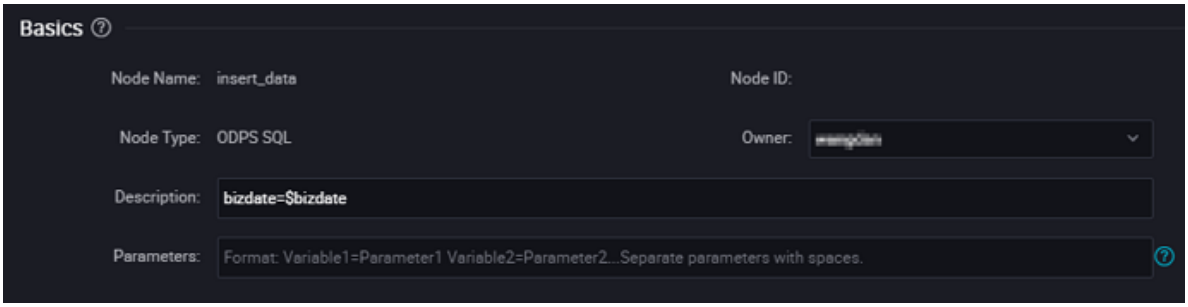
非 **Shell** ノードでは、**${variable name}** (関数が参照されることを示します) をコードに追加します。その後、スケジューリングパラメーターへ割り当てる特定の値を入力します。

たとえば、**ODPS SQL** ノードの場合、**${variable name}** をコードへ追加し、ノードのパラメーター項目 **"variable name =built-in scheduling parameter"** を設定します。

1. コードで参照されているパラメーターでは、スケジューリング中に解決済みの値を追加する必要があります。



2. 値は、コードで参照されている変数へ割り当てる必要があります。 値の割り当て規則は、**variable name** = **parameter** です。



Shell ノードのスケジューリングパラメーターの設定

Shell ノードのパラメーター設定手順は、非 **Shell** ノードの設定手順と似ていますが、規則は異なります。 **Shell** ノードでは、変数名をカスタマイズすることはできません。変数名は、**"$1、$2、$3..."** にする必要があります。

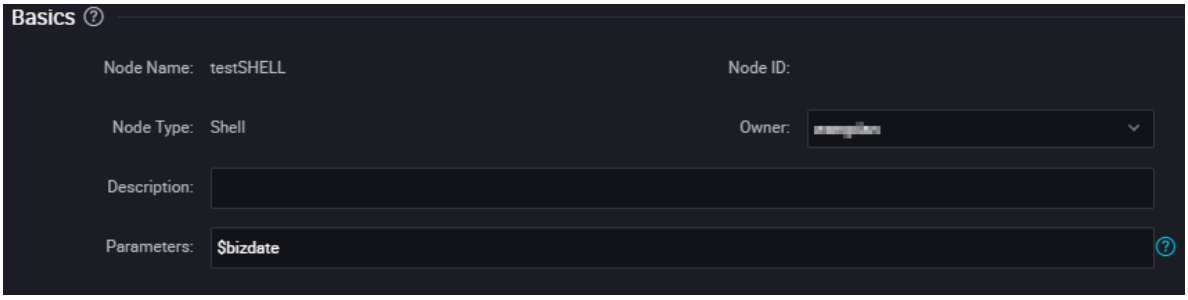
たとえば、**Shell** ノードの場合、コード内の **Shell** 構文宣言は、**$1** です。スケジューリングのノードパラメーターの設定は、**$xxx** (ビルトインスケジューリングパラメーター) です。 つまり、コード内の **$1** を置換するため、**$xxx** の値が使用されます。

1. コードで参照されているパラメーターでは、スケジューリング中に解決済みの値を追加する必要があります。

注：

Shell コードでは、パラメーターの数が **10** になる場合、**${10}** を使って変数を宣言する必要があります。

2. 値は、コードで参照されている変数へ割り当てる必要があります。 値の割り当て規則は、パラメーター 1、パラメーター 2、パラメーター 3 ... (置換された変数は、パラメーターの位置に基づいて解決されます。たとえば、$1 はパラメーター 1 に解決されます。



変数値が固定値の場合

SQL ノードを例にします。 コードの **${variable name}** では、ノードのパラメーター項目 **" variable name = "fixed value""** を設定します。

コード**: select xxxxxx type='${type}'**

スケジューリング変数に割り当てられる値**: type="aaa"**

スケジューリング中、コードの変数は、**type="aaa"** に置換されます。

変数値がビルトインスケジューリングパラメーターの場合

SQL ノードを例にします。 コード内の **${variable name}** では、ノードのパラメーター項目 **" variable name=scheduling parameter""** を設定します。

コード**: select xxxxxx dt=${datetime}**

スケジューリング変数に割り当てられる値**: datetime=$bizdate**

スケジューリング中、今日が**2017** 年 **7** 月 **22** 日の場合、コードの変数は **dt=20170721** に置換されます。

ビルトインスケジューリングパラメーターの一覧

$bizdate: 形式 **yyyymmdd** で示す営業日 注記**:** このパラメーターは広く使用され、ルーチンスケジューリングではデフォルトでは前日に日付となります。

たとえば、**ODPS SQL** ノードのコードでは、**pt=${datetime}** です。 ノードのパラメーター設定では、**datetime=$bizdate** です。 今日が **2017** 年 **7** 月 **22** 日とします。 ノードが今日実行される場合、**$bizdate** は **pt=20170721** に置換されます。

たとえば、**ODPS SQL** ノードのコードでは、**pt=${datetime}** です。 ノードのパラメーター設定では、**datetime=$gmtdate** です。 今日が **2017** 年 **7** 月 **22** 日とします。 ノードが今日実行される場合、**$gmtdate** は **pt=20170722** に置換されます。

$cyctime: タスクのスケジュール時間 毎日のタスクに対し、時間がスケジュールされていない場合、**cyctime** は現在の日の **00:00** になります。 時間の時、分、秒は正確で、これらは通常、時間単位または分単位のスケジューリングタスクに使用されます。 例**: cyctime=$cyctime**

注 **:**

$[] と **${}** を使って設定された時間パラメーターの違いに注意してください。 **$bizdate:** 営業日です。デフォルトでは、現在より **1** 日前の日です。 **$cyctime:** タスクのスケジュールされた時間です。 毎日のタスクの対し、時間がスケジュールされていない場合、タスクは現在の日の **00:00** に実行されます。 時間の時、分、秒は正確で、これらは通常、時間単位または分単位のスケジューリングタスクに使用されます。 たとえば、タスクが現在の日の **00:30** に実行されるようにスケジュールされている場合、スケジュール時間は、**yyyy-mm-dd 00:30:00** です。 **[]** を使って時間パラメーターを設定した場合、**cyctime** は実行のベンチマークとして使用されます。 この使用の詳細については、次の手順をご参照ください。 時間の計算方法は、**Oracle** と同じです。 データ作成中、パラメーターは選択した営業日プラス **1** 日に置換されます。 たとえば、営業日 **20140510** がデータ作成中に選択された場合、**cyctime** は **20140511** に置換されます。

$jobid: タスクが属しているワークフローの **ID** 例**: jobid=$jobid**

$nodeid: ノードの **ID** 例**: nodeid=$nodeid**

$taskid: タスクの **ID**、つまりノードインスタンスの **ID** です。 例**: taskid=$taskid**

$bizmonth: 形式 **yyyymm** で示す営業月

- 営業日の月が現在の月と同じ場合、**$bizmonth** = 営業日の月マイナス **1** です。同じでない場合は、**$bizmonth** = 営業日の月です。
- たとえば、**ODPS SQL** ノードのコードでは、**pt=${datetime}** です。 ノードのパラメーター設定は、**datetime=$bizmonth** です。 今日が **2017** 年 **7** 月 **22** 日とします。 ノードが今日実行される場合、**$bizmonth** は **pt=201706** に置換されます。

$gmtdate: 形式 **yyyymmdd** で示される現在の日付です。 このパラメーターの値は、デフォルトでは現在の日付です。 データ作成中、入力である **gmtdate** は営業日プラス **1** です。

カスタムパラメーター **${…}** パラメーターの説明**:**

- **$bizdate** に基づいてカスタムした時間形式では、**yyyy** は **4** 桁の年、**yy** は **2** 桁の月、**mm** は月、**dd** は日を示します。 パラメーターを組み合わせることができます。たとえば、**${yyyy}**、 **${yyyymm}**、 **${yyyymmdd}** および **${yyyy-mm-dd}** です。
- **$bizdate** は年、月、日です。 カスタムパラメーター **${……}** でも、年、月、日のみを表現します。

・ 特定の期間の前後の範囲を取得する方法

次の N 年: **${yyyy+N}**

前の N 年: **${yyyy-N}**

次の N 月: **${yyyymm+N}**

前の N 月: **${yyyymm-N}**

次の N 週間: **${yyyymmdd+7*N}**

前の N 週間: **${yyyymmdd-7*N}**

次の N 日: **${yyyymmdd+N}**

前の N 日: **${yyyymmdd-N}**

${yyyymmdd}:** 形式 **yyyymmdd** で示す営業日です。 値は、**$bizdate の値と一致します。

・ このパラメーターは広く使用され、ルーチンスケジューリングではデフォルトで、前日の日付です。 このパラメーターの形式をカスタムすることができます。たとえば、**${yyyy-mm-dd}** の形式を **yyyy-mm-dd** にカスタムできます。
・ たとえば、**ODPS SQL** ノードのコードでは、**pt=${datetime}** です。 ノードのパラメーター設定では、**datetime=${yyyymmdd}** です。 今日が **2013** 年 **7** 月 **22** 日とします。 ノードが今日実行される場合、**${yyyymmdd}** は **pt=20130721** に置換されます。

${yyyymmdd-/+N}: yyyymmdd プラス、またはマイナス **N** 日

${yyyymm-/+N}: yyyymm プラス、またはマイナス **N** 月

${yyyy-/+N}: 年 **(yyyy)** プラス、またはマイナス **N** 年

${yy-/+N}: 年 **(yy)** プラス、またはマイナス **N** 年

yyyymmdd は営業日を示し、**yyyy-mm-dd** のように区切ることができます。 上記のパラメーターでは、営業日の年、月、日から取得されます。

例:

・ **ODPS SQL** ノードのコードでは、 **pt=${datetime}** です。 ノードのパラメーター設定では、**datetime=${yyyy-mm-dd}** です。 今日が **2018** 年 **7** 月 **22** 日とします。 ノードが今日実行される場合、**${yyyy-mm-dd}** は **pt=2018-07-21** に置換されます。
・ **ODPS SQL** ノードのコードでは、 **pt=${datetime}** です。 ノードのパラメーター設定では、**datetime=${yyyymmdd-2}** です。 今日が **2018** 年 **7** 月 **22** 日とします。 ノードが今日実行される場合、**${yyyymmdd-2}** は **pt=20180719** に置換されます。

- **ODPS SQL** ノードのコードでは、 **pt=${datetime}** です。 ノードのパラメーター設定では、**datetime=${yyyymm-2}** です。 今日が **2018** 年 **7** 月 **22** 日とします。 ノードが今日実行される場合は、**${yyyymm-2}** は **pt=201805** に置換されます。
- **ODPS SQL** ノードのコードでは、 **pt=${datetime}** です。 ノードのパラメーター設定では、**datetime=${yyyy-2}** です。 今日が **2018** 年 **7** 月 **22** 日とします。 ノードが今日実行される場合は、**${yyyy-2}** は **pt=2018** に置換されます。

ODPS SQL ノード設定では、複数のパラメーターに値が割り当てられます。たとえば、**startdatetime=$bizdate enddatetime=${yyyymmdd+1} starttime=${yyyy-mm-dd} endtime=${yyyy-mm-dd+1}** です。

例**: ($cyctime=20140515103000** とします**)**

- **$[yyyy] = 2014**、**$[yy] = 14**、 **$[mm] = 05, $[dd] = 15**、 **$[yyyy-mm-dd] = 2014-05-15**、 **$[hh24:mi:ss] = 10:30:00**、 **$[yyyy-mm-dd hh24:mi:ss] = 2014-05-1510:30:00**
- **$[hh24:mi:ss - 1/24] = 09:30:00**
- **$[yyyy-mm-dd hh24:mi:ss -1/24/60] = 2014-05-1510:29:00**
- **$[yyyy-mm-dd hh24:mi:ss -1/24] = 2014-05-1509:30:00**
- **$[add_months(yyyymmdd,-1)] = 2014-04-15**
- **$[add_months(yyyymmdd,-12*1)] = 2013-05-15**
- **$[hh24] =10**
- **$[mi] =30**

パラメーター **$cyctime** のテスト方法

インスタンスを実行した後、ノードを右クリックし、**[check the node attribute]** を行います。 スケジュール時間が、インスタンスを定期的に実行する時間になっているかどうかを確認します。

パラメーター値の後の結果は、スケジュール時間マイナス **1** 時間に設定されます。

よくある質問

- **Q:** テーブルパーティションの形式が **pt=yyyy-mm-dd hh24:mi:ss** ですが、スケジューリングパラメーターではスペースが使えません。 **$[yyyy-mm-dd hh24:mi:ss]** の形式で設定した方が良いですか。

  A: カスタム変数パラメーター **datetime=$[yyyy-mm-dd]** と **hour=$[hh24:mi:ss]** を使ってそれぞれの日と時間を取得します。 その後、コードで結合して **pt="${datetime} ${hour}"** を作成します。 **(2** つのカスタムパラメーターはスペースで区切ります。**)**

- **Q:** コードのテーブルパーティションが **pt="${datetime} ${hour}"** です。 実行中の最終時間のデータを取得するには、カスタム変数パラメーター **datetime=$[yyyymmdd]** と **hour=$[hh24-1/24]** を使ってそれぞれの日と時間を取得することができます。 ただし、インスタンスの実行が **00:00** の場合、計算結果は前日の **23:00** ではなく現在の日の **23:00** になってしまいます。 この場合、どのような処置をとることができますか。

  A: datatime の形式を **$[yyyymmdd-1/24]** に変更し、時間の形式を **$[hh24-1/24]** のままにします。 計算結果は次のとおりとなります。

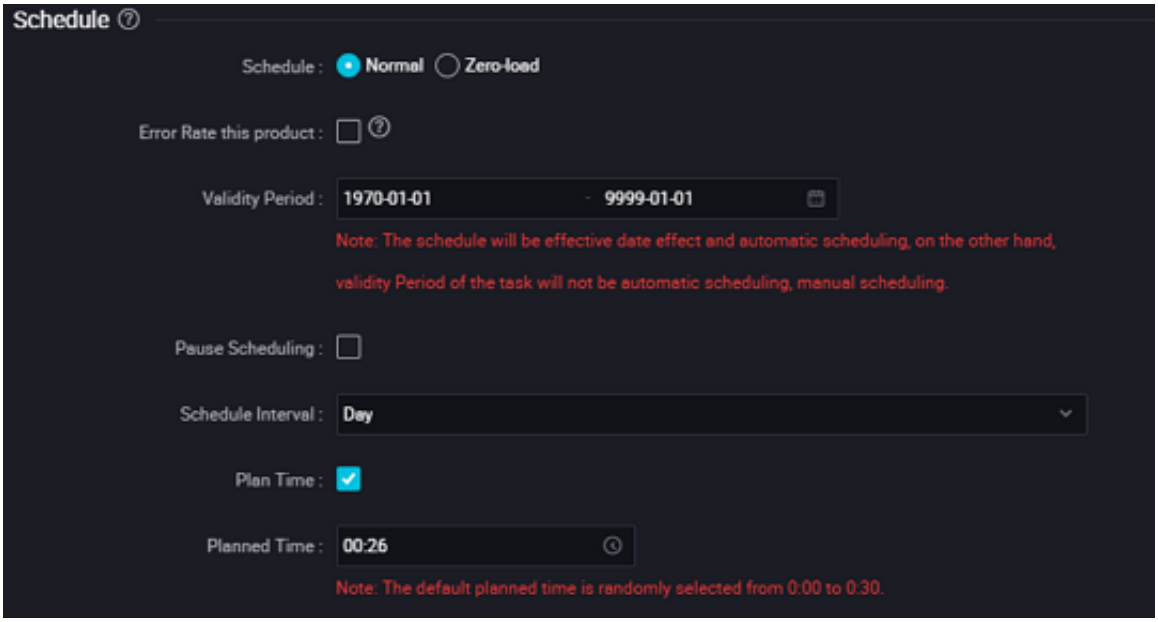
  - インスタンスのスケジュール時間が **2015-10-27 00:00:00** の場合、**$[yyyymmdd-1/24]** の値と **$[hh24-1/24]** の値はそれぞれ、**20151026** と **23** になります。これは、スケジュール時間マイナス **1** 時間が、昨日に属する時間値だからです。
  - インスタンスのスケジュール時間が **2015-10-27 01:00:00** の場合、**$[yyyymmdd-1/24]** の値と **$[hh24-1/24]** はそれぞれ、**20151027** と **00** になります。これは、スケジュール時間マイナス **1** 時間が、現在の日に属する時間値だからです。

DataWorks では **4** つの実行方法が提供されています。

- データ開発ページでの実行**:** 正常に実行するためには、パラメーター設定ページで一時的な値の割り当てが必要になります。 しかし、割り当て値はタスク属性としては保存されず、他 **3** つの実行モードに影響を与えません。
- 任意の周期での自動実行**:** パラメーター編集ボックスで設定する必要はありません。パラメーターは、現行インスタンスでスケジュールされたランタイムへ自動的に置換されます。
- テスト実行/データサプリメント実行**:** 実行をトリガーする営業日を設定する必要があります。各インスタンスのシステムパラメーター値 **2** つを取得するため、前述の数式によってスケジュールされるランタイムが求められます。

1.5.3 時間属性

時間属性設定ページは次の図で示すとおりです。



ノード状態

- **Normal:** 次のスケジューリングサイクルに基づいてノードがスケジュールされています。 このオプションはデフォルトで選択されています。
- **Zero-load:** このオプションを選択すると、次のスケジューリングサイクルに基づいてノードが設定されスケジュールされます。 しかし、タスクがスケジュールされると、タスクを実行せずに成功結果が直接返されます。
- **Error retries:** ノードにエラーが発生した場合、ノードを再実行することができます。 デフォルトのエラーでは自動的に **2** 分間隔で **3** 回再実行されます。
- **Suspend scheduling:** このチェックボックスを選択すると、次のスケジューリングサイクルに基づいてノードが設定されスケジュールされます。 しかし、このタスクがスケジュールされると、タスクを実行せずに失敗結果が直接返されます。 タスクが中断されているが後で実行する場合に使用します。

スケジューリング間隔

DataWorks では、タスクが正常に送信されると、下にあるスケジューリングシステムはタスクの時間属性に基づいて、次の日から毎日インスタンスを生成します。実行結果と依存関係にあるアップストリームインスタンスのタイムポイントに基づいてインスタンスを実行します。 **23:30** 以降に正常に送信されたタスクの場合、インスタンスは **3** 日目から生成されます。

注：

タスクを毎週月曜日に実行する必要がある場合、ランタイムが 月曜日に設定されている場合のみタスクが実行されます。 ランタイムが月曜日に設定されていない場合 、タスク (直接成功と設

定されている) が偽りで実行されます。 この理由から、テスト実行またはデータサプリメント実行で週単位のスケジュールタスクでは営業日 = ランタイム - 1 を設定します。

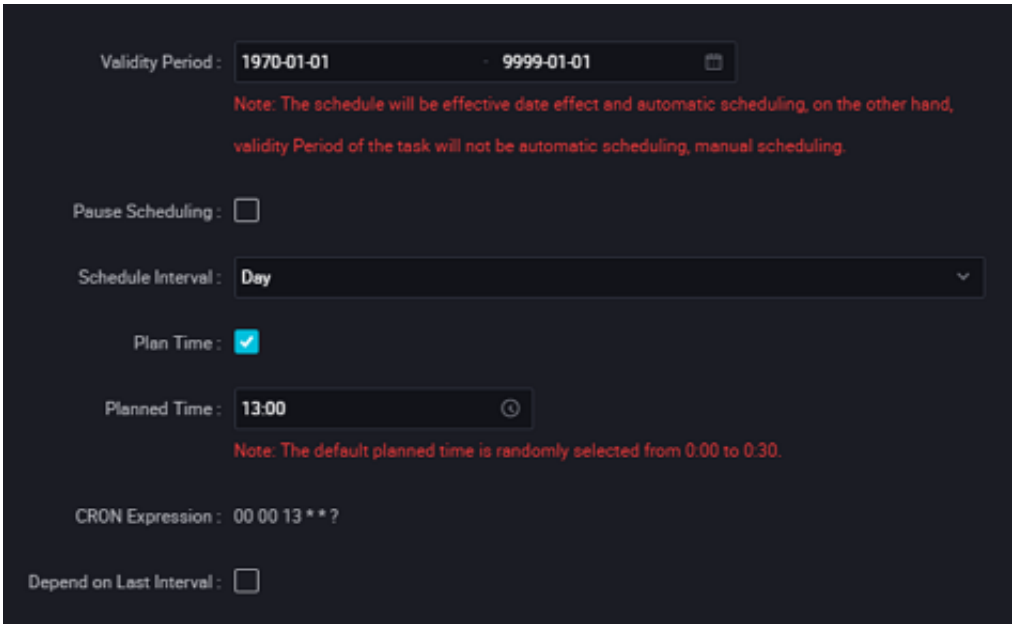
サイクル通りに実行されるタスクでは、依存関係の優先順位は、時間属性の優先順位よりも高くなります。 したがって、時間属性で指定した時間になると、タスクインスタンスはすぐに実行されず、まずすべてのアップストリームインスタンスが正常に実行されたかどうかを確認します。

- 依存関係にある上位インスタンスすべてが正常に実行されておらず、スケジュールされたランタイムになっている場合、インスタンスは実行以外の状態を維持します。
- 依存関係にある上位インスタンスすべてが正常に実行されておらず、スケジュールされたランタイムになっている場合、インスタンスは実行以外の状態を維持します。
- 依存関係にある上位インスタンスすべてが正常に実行されており、スケジュールされたランタイムになっている場合、インスタンスはリソース待ち状態となり、実行準備完了状態となります。

日単位のスケジューリング

日単位でスケジュールされたタスクは、毎日 1 回自動的に実行されます。 周期タスクを作成すると、デフォルトでタスクは毎日 00:00 に実行されるように設定されます。 必要に応じて、別のランタイムを指定することができます。 たとえば、次の図に示すとおり、ランタイムを毎日 13:00 に指定することができます。

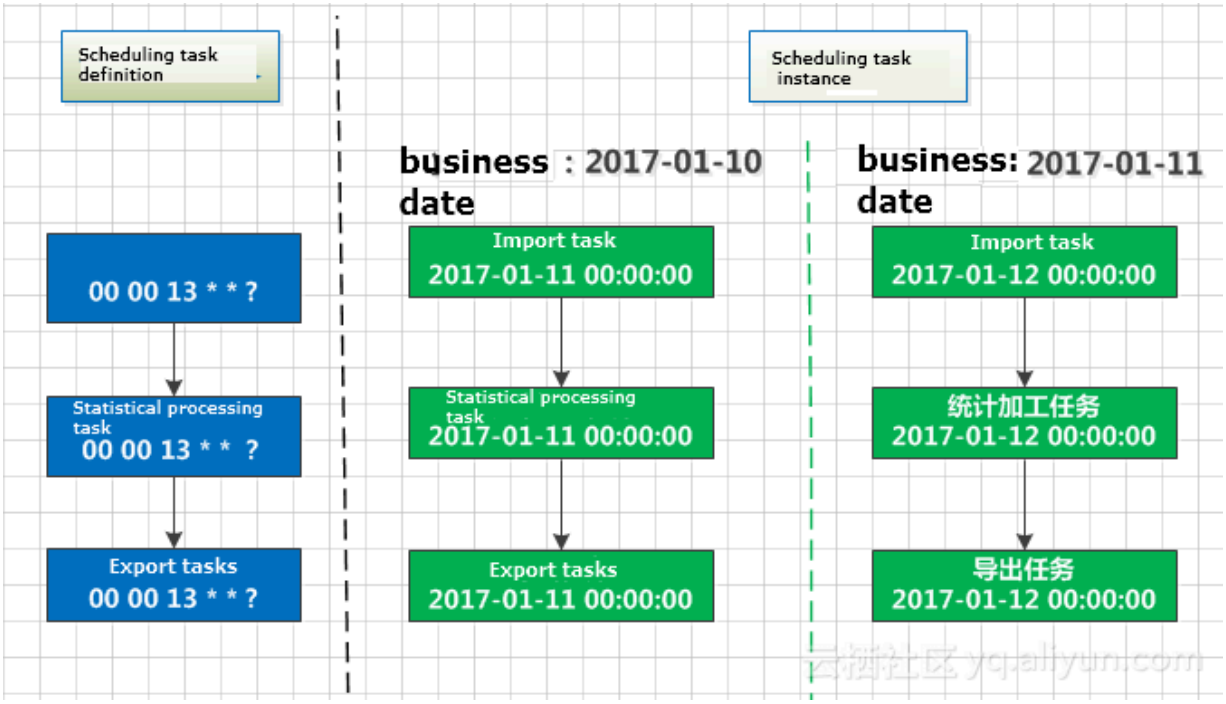
1. Regular Scheduling の選択が解除されている場合、毎日のタスクインスタンスのスケジュール時間は、YYYY-MM-DD の現在の日付となります。0:00 から 0:30 の間でランダムに生成されるデフォルトのスケジューリング時間となります。
2. Regular Scheduling が選択されている場合、毎日のタスクインスタンスのスケジュール時間は、YYYY-MM-DD の現在の日付となり、スケジュール時間は HH:MM:SS となります。スケジュールされたタスクは、上位タスクが正常に実行され、スケジュールされた時間になった時のみ実行されます。 この条件のどちらかが満たされていない場合、タスクを実行することはできません。 条件に順序はありません。



使用例:

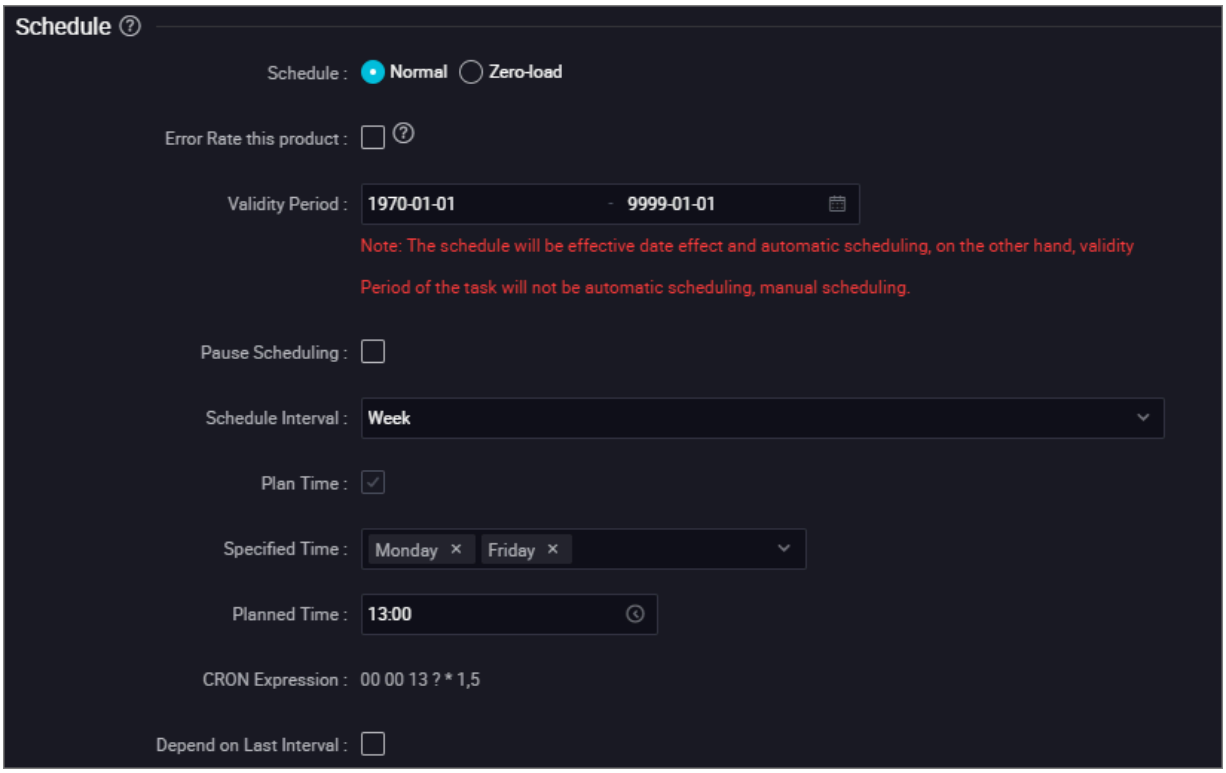
インポート、統計処理、エクスポートタスクは、上記の図に示すとおり、すべて毎日ランタイム **13:00** に行われるタスクです。 統計処理タスクはインポートタスクに依存し、エクスポートタスクは統計処理タスクに依存します。 次の図では、これらの依存関係 (統計処理タスクの依存属性設定、インポートタスクに設定されているアップストリームタスク) を示します。

上記の図にある設定に基づいて、スケジューリングシステムは、タスクのインスタンスを自動的に生成し、次のとおり実行します。



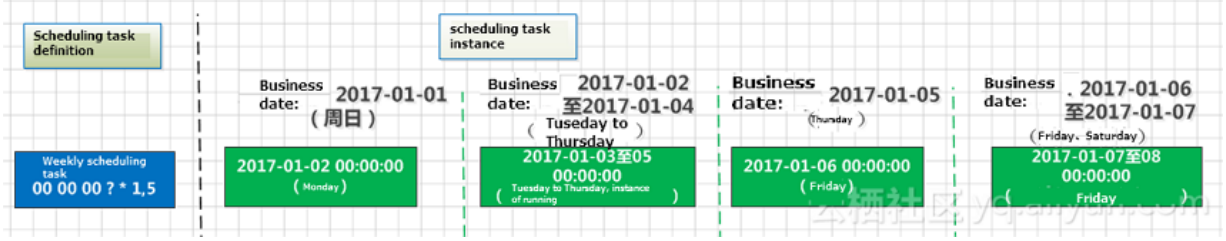
週単位のスケジューリング

週単位でスケジュールされているタスクは、毎週特定の日の特定の時間に自動的に実行されます。 特定されていない日に到達した場合、システムはインスタンスを生成します。また、論理の実行を行ったりリソースを消費したりせずに、直接インスタンスを正常に実行された状態と設定し、ダウンストリームインスタンスが適切に実行されるようにします。



上記の図に示すとおり、毎週月曜日と金曜日に生成されたインスタンスはスケジュールどおり実行され、毎週火曜日、水曜日、木曜日、土曜日、日曜日に生成されたその他のインスタンスは、正常に実行されたと設定されます。

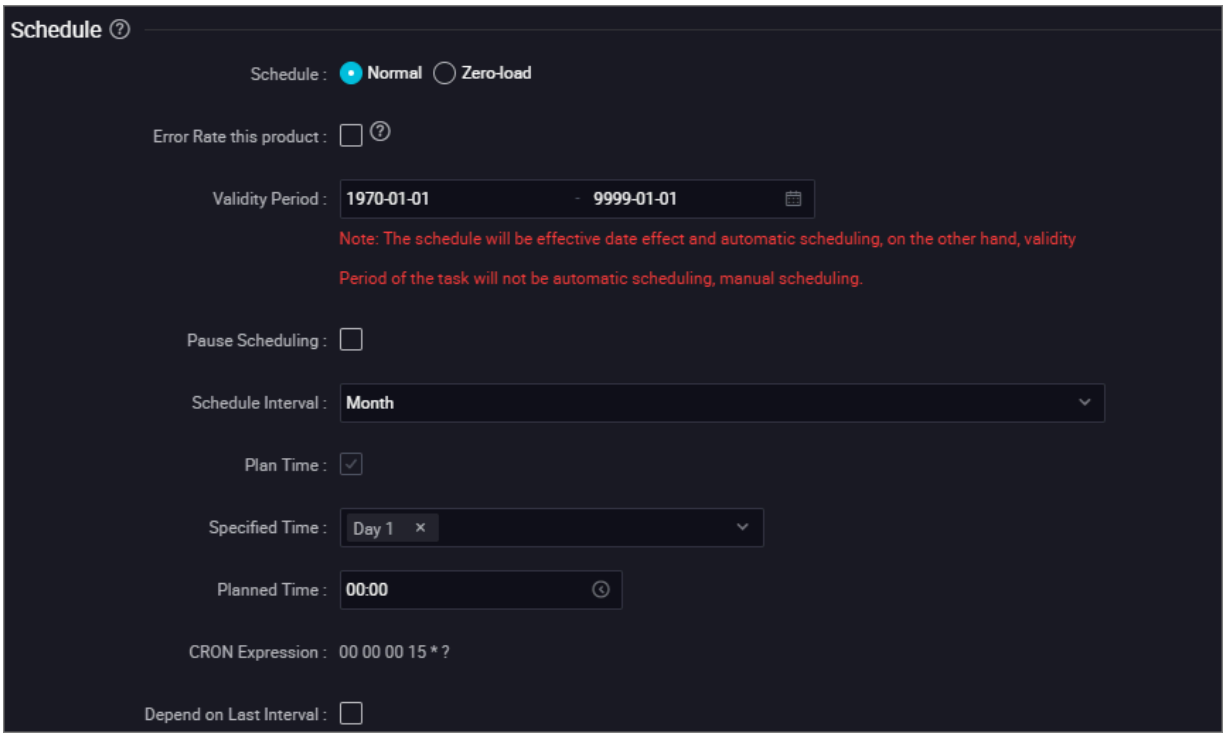
上記の図にある設定に基づいて、スケジューリングシステムは、タスクのインスタンスを自動的に生成し、次のとおり実行します。



月単位のスケジューリング

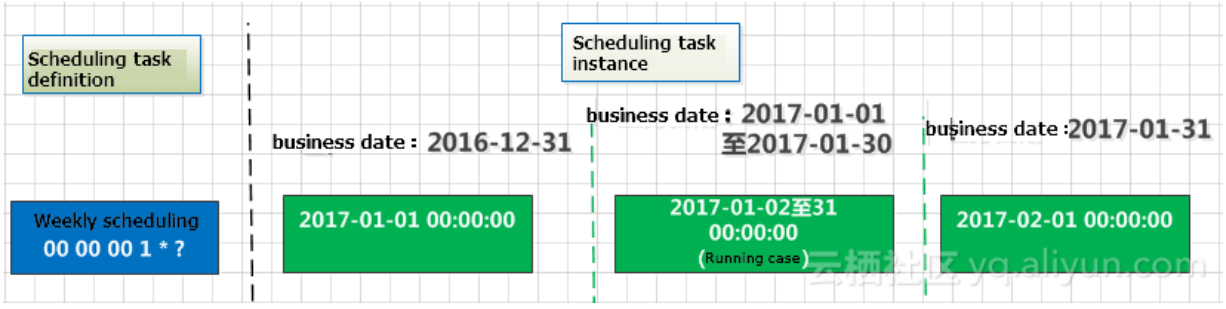
月単位でスケジュールされているタスクは、毎月特定の日の特定の時間に自動的に実行されます。 特定されていない日に到達した場合、システムはインスタンスを毎日作成します。また、論

理の実行を行ったりリソースを消費したりせずに、直接インスタンスを正常に実行された状態と設定し、下位インスタンスが適切に実行されるようにします。



上記の図に示すとおり、毎月 1 日に生成されたインスタンスはスケジュールどおりに実行され、その月の残りの日に生成されたインスタンスは、正常に実行されたと設定されます。

上記の図にある設定に基づいて、スケジューリングシステムは、タスクのインスタンスを自動的に生成し、次のとおり実行します。

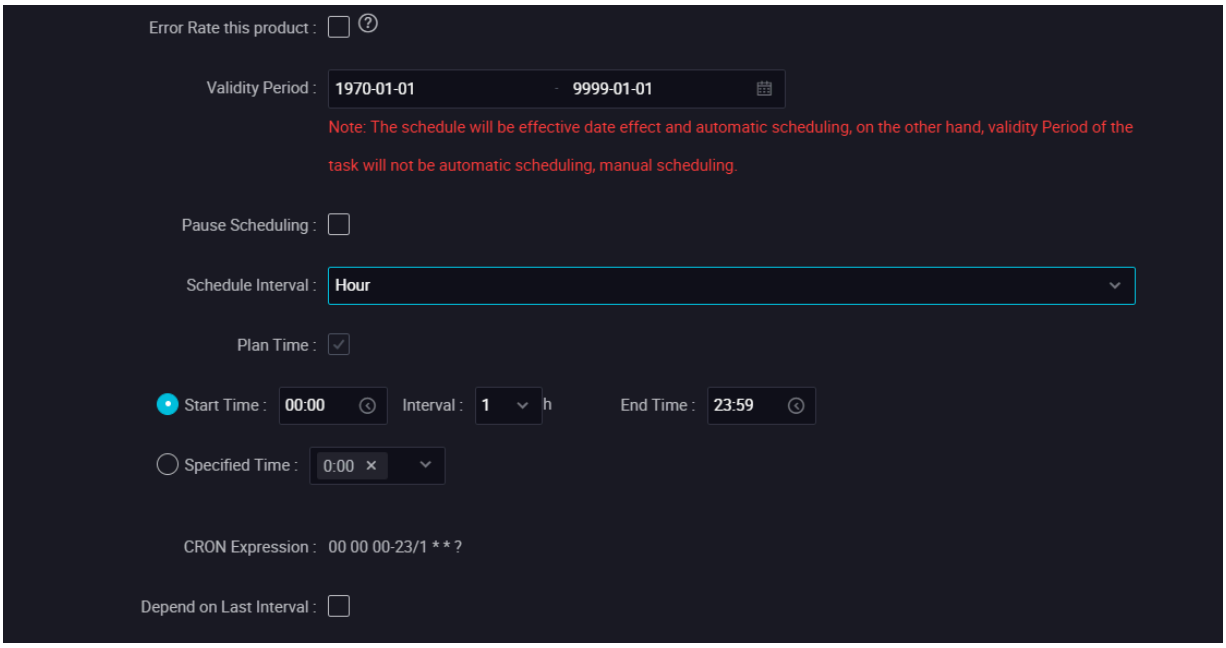


時間単位のスケジューリング

時間単位でスケジュールされたタスクは、毎日 **Nx1** 時間ごとに実行されます。たとえば、毎日 **1:00** から **4:00** までの間、**1** 時間ごとに実行されます。

> 注：
> 実行間隔は、左クローズと右クローズ原理に基づいて計算されます。 たとえば、時間単位でスケジュールされたタスクは、**0:00** から **3:00** までの間、**1** 時間ごとに実行されるように設定さ

れている場合、時間の間隔は **[0:00, 3:00]** で、間隔は **1** 時間です。 スケジューリングシステムは、毎日インスタンスを **4** つ生成し、**0:00**、**1:00**、**2:00**、**3:00** に実行します。

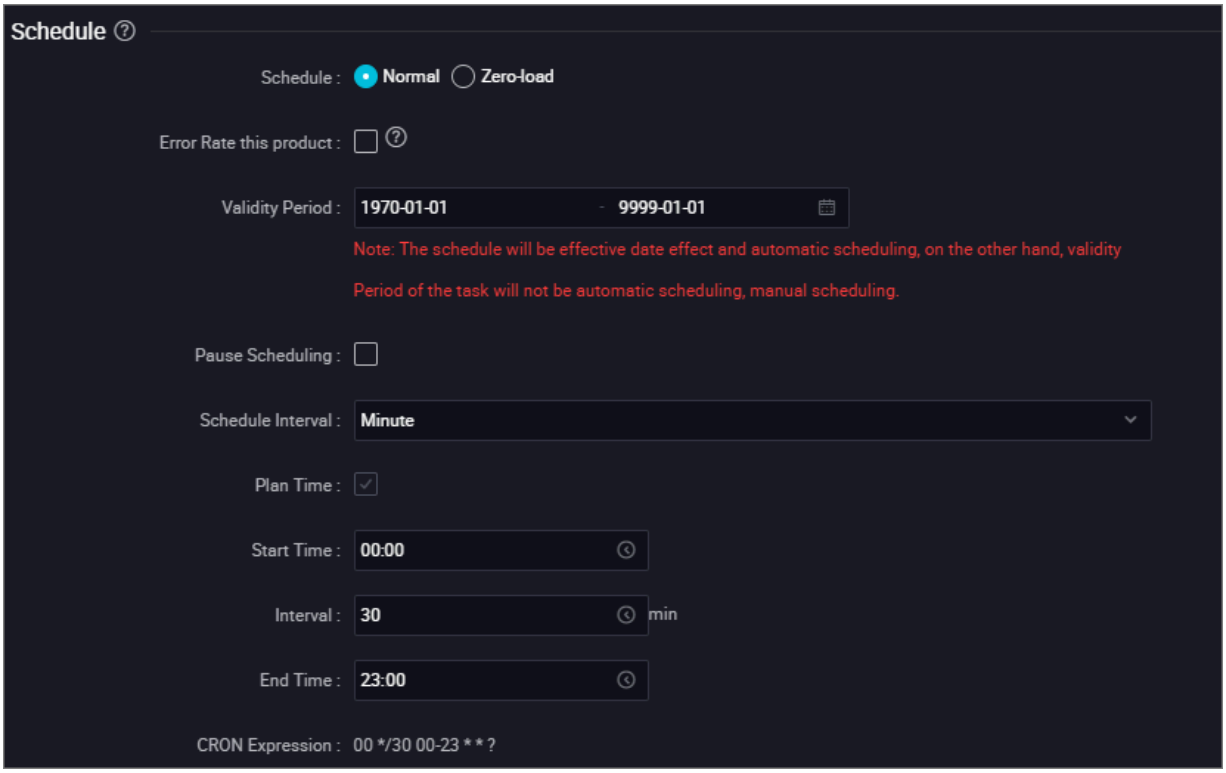


上記の図に示すとおり、自動スケジューリングは毎日 **0:00** から **23:59** の間の **6** 時間ごとにトリガーされます。 したがって、スケジューリングシステムは、次のとおり、タスクのインスタンスを自動的に生成し、実行します。

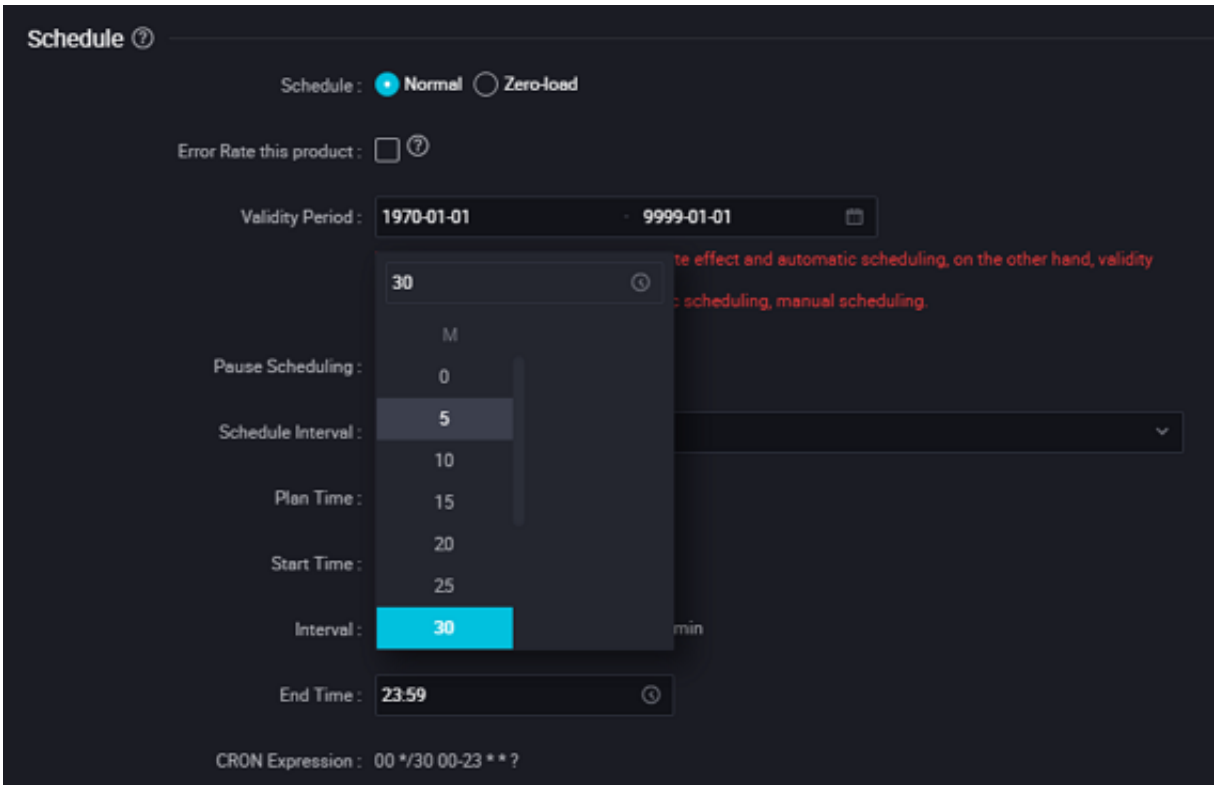
分単位のスケジューリング

分単位でスケジュールされたタスクは、次の図に示すとおり、毎日 **N x 1** 分 ごとに実行されます。

タスクは **0:00** から **23:00** までの間、**30** 分ごとにスケジュールされています。



現在、分単位のスケジューリングは、最低 5 分単位での設定をサポートしています。 時間の表現は選択式で、編集することはできません。



よくある質問

Q: 上位タスク **A** が時間単位でスケジュールされ、下位タスク **B** が日単位でスケジュールされています。さらに毎日タスク **A** が完了した後にタスク **B** を実行する必要がある場合、タスク **A** とタスク **B** はお互いに依存関係にありますか。

A: 日単位のタスクは、時間単位のタスクに依存します。 タスク **A** が時間単位でスケジュールされている場合、日単位でスケジュールされているタスク **B** は不定期にスケジュールされたものとなります。タスク **A** とタスク **B** はお互いに依存関係にあります。タスク **B** は毎日 **24** 時間タスク **A** が正常にインスタンスを実行した後に実行されます。 (依存関係の設定に関する詳細は、スケジューリング依存関係の説明をご参照ください。) したがって、各周期タスクはそれぞれに依存しており、各タスクのスケジューリングサイクルは、タスクの時間属性によって判断されます。

Q: タスク **A** を毎時間実行し、タスク **B** は **1** 日 **1** 回実行し、さらにタスク **A** が **1** 回目に正常に実行された後にタスク **B** が開始する場合、 どのように設定すれば良いですか。

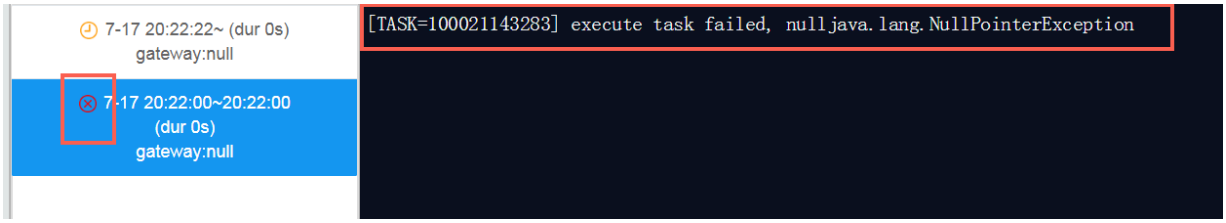
A: タスク **A** を設定する際、**[Previous Cycle Dependent]** と **[Current Node]** を選択し、タスク **B** のスケジュール時間を **0:00** に設定します。 このように、自動的に毎日スケジュールされるインスタンスでタスク **B** のインスタンスは、タスク **A** の **0:00** インスタンスのみ、つまりタスク **A** の **1** つ目のインスタンスにのみ依存することとなります。

Q: タスク **A** を毎週月曜日に実行し、タスク **B** はタスク **A** に依存させる場合、タスク **B** が毎週月曜日に実行されるよう、どのように設定すれば良いですか。

A: タスク **B** の時間属性をタスク **A** の時間属性と同じように設定します。スケジューリングサイクルを **[Weekly]** と **[Monday]** に設定します。

Q: タスクのインスタンスは、タスクを削除した場合、影響をうけますか。

A: ある期間実行した後にタスクを削除した場合、そのインスタンスは残ります。これはスケジューリングシステムが時間属性に基づいて、**1** つ以上のタスクのインスタンスをまだ生成しているからです。このため、タスクが削除された後にインスタンスがトリガーされた場合、必要とされるコードが見つからないため、次のエラーメッセージが表示されます。



Q: 毎月末日に毎月のデータを計算する場合、どのようにすれば良いですか。

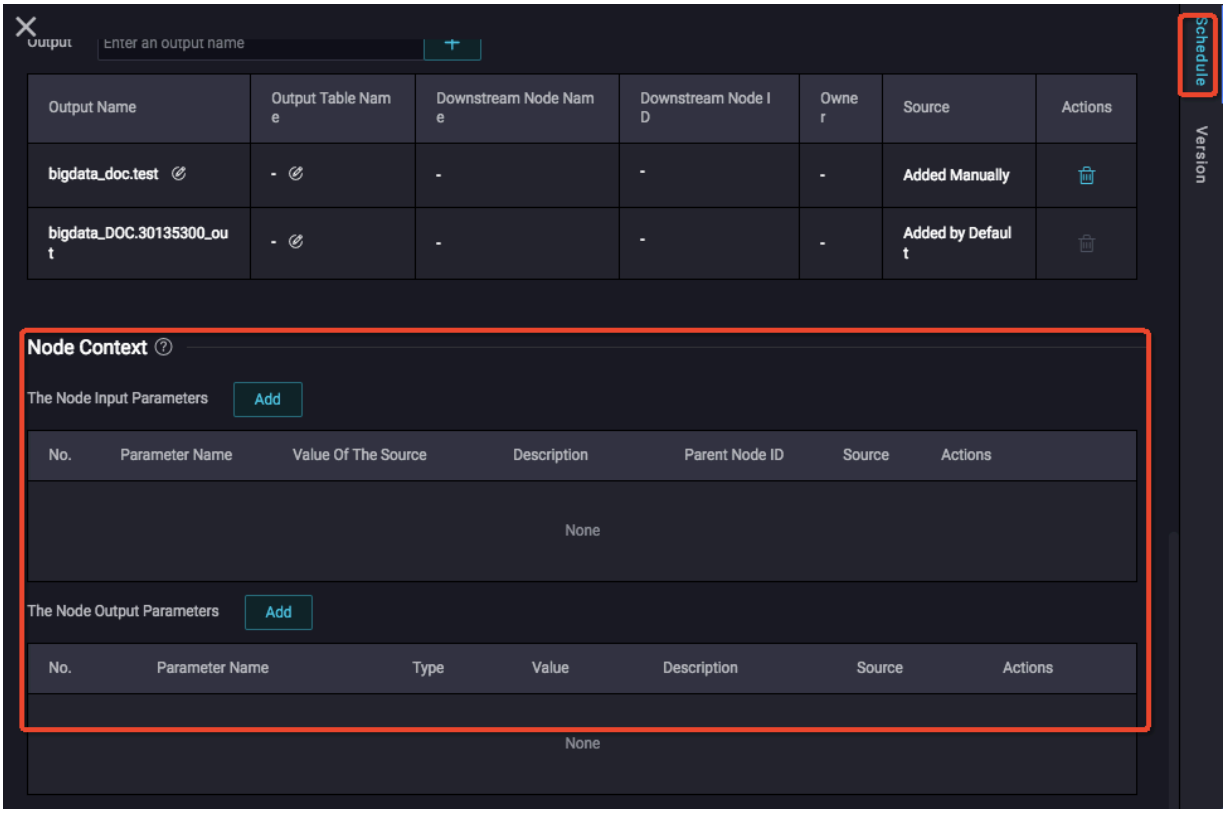
A: 現在、システムでは毎月末日をランタイムとして設定することはできません。 したがって、タスクを毎月 **31** 日に実行するように設定する場合、スケジューリングは **31** 日間ある月には **1** 日トリガーされ、インスタンスが生成されます。他の日のインスタンスは正常に実行されたと設定されます。

月次統計では、毎月 **1** 日に前月のデータを計算することを推奨します。

1.5.4 ノードコンテキスト

ノードコンテキスト **(Node Context)** は、上位ノードと下位ノード間でパラメーターを転送するために使用します。 ノードコンテキスト機能では、最初に上位ノードの出力パラメーターと値を定義し、次に下位ノードの入力パラメーターを定義するのが基本的な使い方です (入力パラメータ値には、上位ノードの出力パラメーターが参照されます)。 下位ノードでこのパラメーターを使用することで、上位ノードから転送される値を取得できます。

ノードコンテキストのパラメーターは、次の図に示すとおり、特定ノードの**[Schedule]** > **[Node Context]** で設定します。



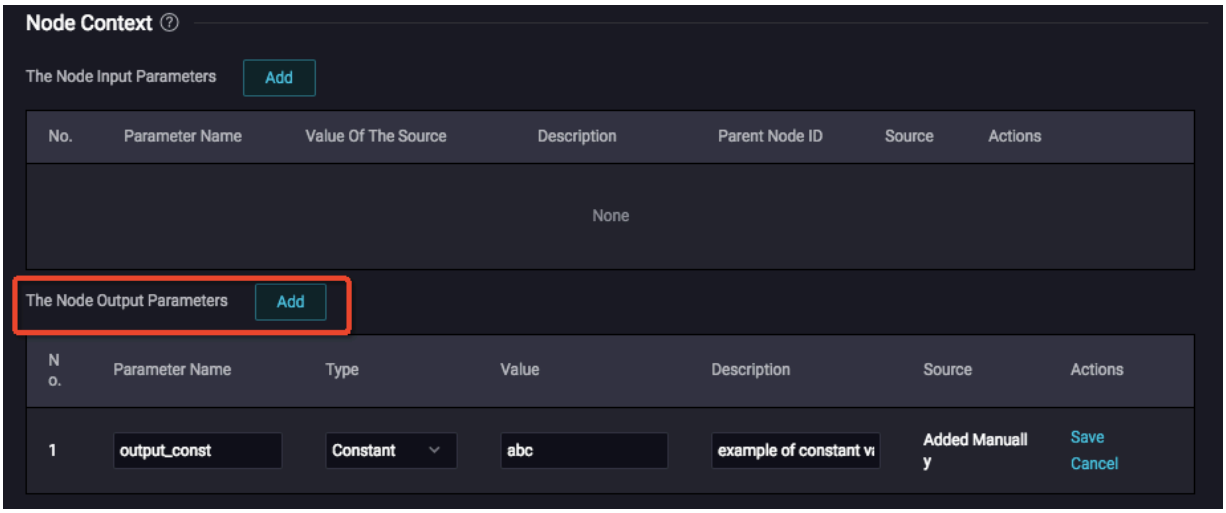
出力パラメーター

[The Node Output Parameters] は**[Node Context]** で定義します。 出力パラメーター値には、「**Constant**」 と 「**Variable**」 の **2** 種類あります。 「**Constant**」 は固定文字列です。 「**Variable**」 はシステムでサポートされているグローバル変数です。 上位ノードで出力パラ

メーターが送信された後、その出力パラメーターを入力パラメーター値として下位ノードで再利用できます。

注：

現行ノード (PyODPS ノードなど) に定義されている出力パラメーターに対して、内部コードを記述して値を割り当てることはできません。



パラメーターは次のように記述します。

フィールド	説明	注意
No.	**[No.]** の値はシステムによって生成され、自動的に加算されます。	**N/A**
Parameter name	出力パラメーター名を定義します。	**N/A**
Type	パラメータータイプです。	出力パラメーター値には、`Constant` と `Variable` の **2** 種類あります。
Value	ソースの値です。	1. **[Type]** で`Constant` が選択されている場合、文字列を直接入力することができます。 2. **[Type]** で `Variable`が選択されている場合、システム変数、スケジュールビルトインパラメーター、カスタマイズパラメーター **$ {...}** と **$ [⋯]** を使用できます。
Description	パラメーターの簡単な説明です。	**N/A**

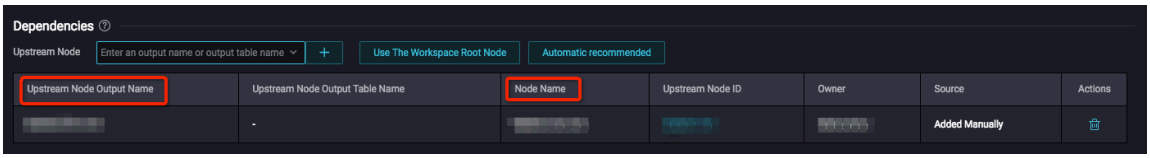
Action	「**Edit**」 および 「**[Delete**」 を選択できます。	「**Edit**」 および 「**Delete**」 は、下位ノードと依存関係がある場合は使用できません。 上位ノードへの参照を追加する前に、上位ノードの出力パラメーターが正しく定義されていることを確認してください。

入力パラメーター

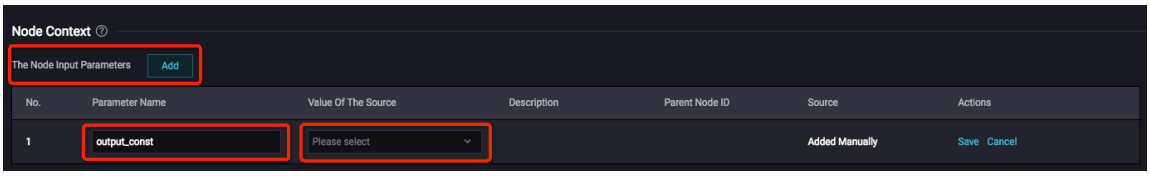
[The Node Input Parameters] は、依存関係のある上位ノードの出力値の参照を定義するために使用します。他のパラメーター同様に、ノード内で使用することができます。

- **[The Node Input Parameters]** の定義

1. **[Dependencies]**で、依存関係のある上位ノードを追加します。



2. **[Node Context]** > **[The Node Input Parameters** を選択し、上位ノードの値を参照する入力パラメーターを定義します。

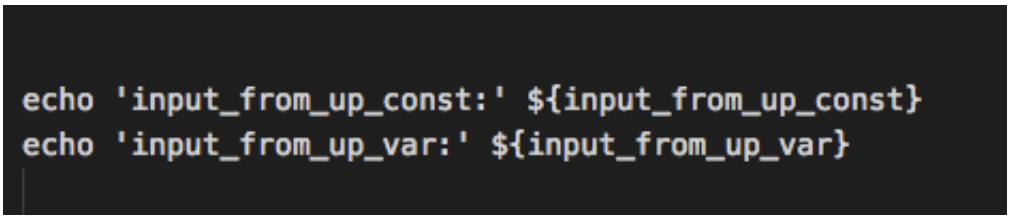


パラメーターは次のように記述します。

フィールド	説明	注意
No.	**[No.]** の値はシステムによって生成され、自動的に加算されます。	**N/A**
Parameter name	入力パラメーター名を定義します。	**N/A**
Value Of the Source	パラメーターの値ソースで、上位ノードの値を参照します。	上位ノードが実行中の特定のパラメーター値です。
Description	パラメーターの簡単な説明です。	上位ノードから自動的に解析されます。
Parent Node ID.	親ノード **ID** です。	上位ノードから自動的に解析されます。
Action	「**Edit**」 および 「**Delete**」 を選択できます。	**N/A**

- 入力パラメーターの使用

  定義された入力パラメーターの再利用する際の形式は、他のシステムと同様です。 形式は、`${input parameter name}` です。 たとえば、**Shell** ノードの参照は次の図に示すとおりです。



サポート対象のグローバル変数

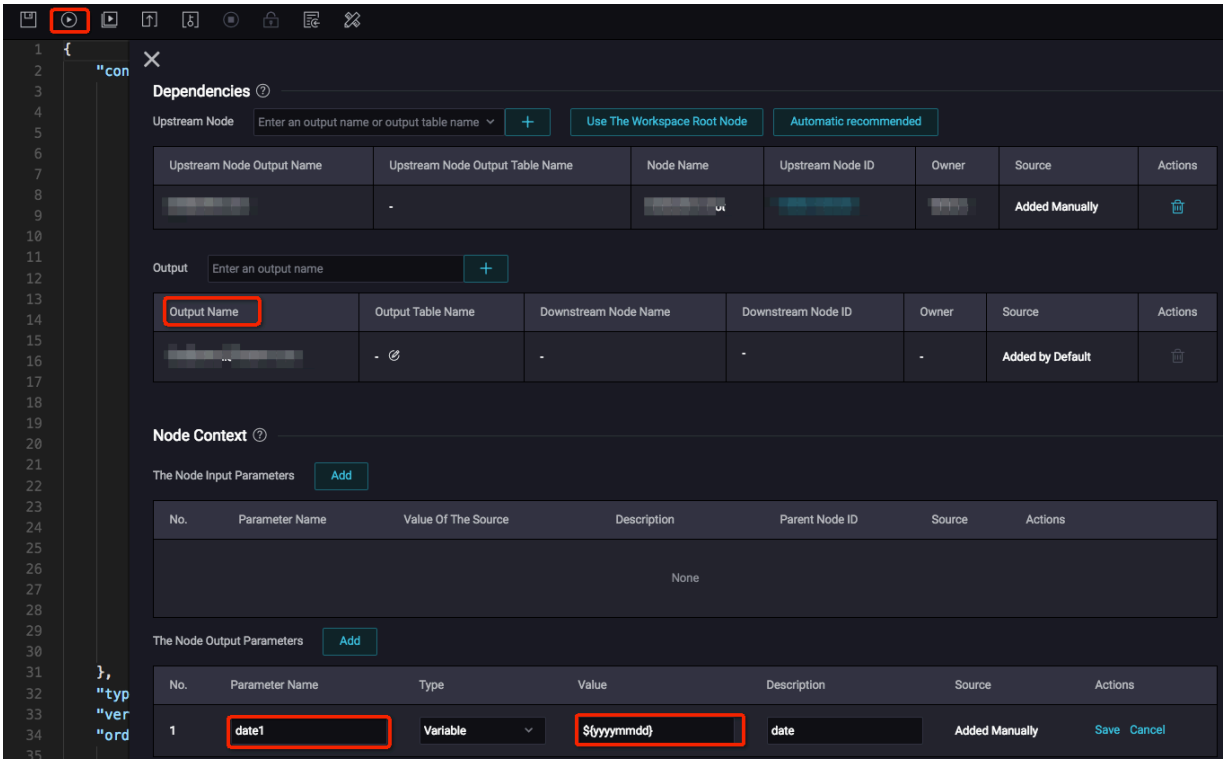
- システム変数

```
$ {projectid}: Project ID
$ {project name}: MaxCompute project name
$ {nodeid}: Node ID
$ {gmtdate}: 00:00:00 at the instance date,format: 'yyyy-mm-dd 00:00
:00'.
$ {taskid}: Instance Task ID
$ {seq}: Task instance sequence number,represents the instance's
sequence number in the same node on current day.
$ {cyctime}: instance time
$ {status}: Status of instance-Success, Failure
$ {bizdate}: Business Date
$ {finishtime}: Instance End Time
$ {taskType}: Instance Status——NORMAL,MANUAL,PAUSE,SKIP,UNCHOOSE,
SKIP_CYCLE
$ {nodeName}: Node name
```

- パラメーター設定の詳細は、「パラメーターの設定」をご参照ください。

例

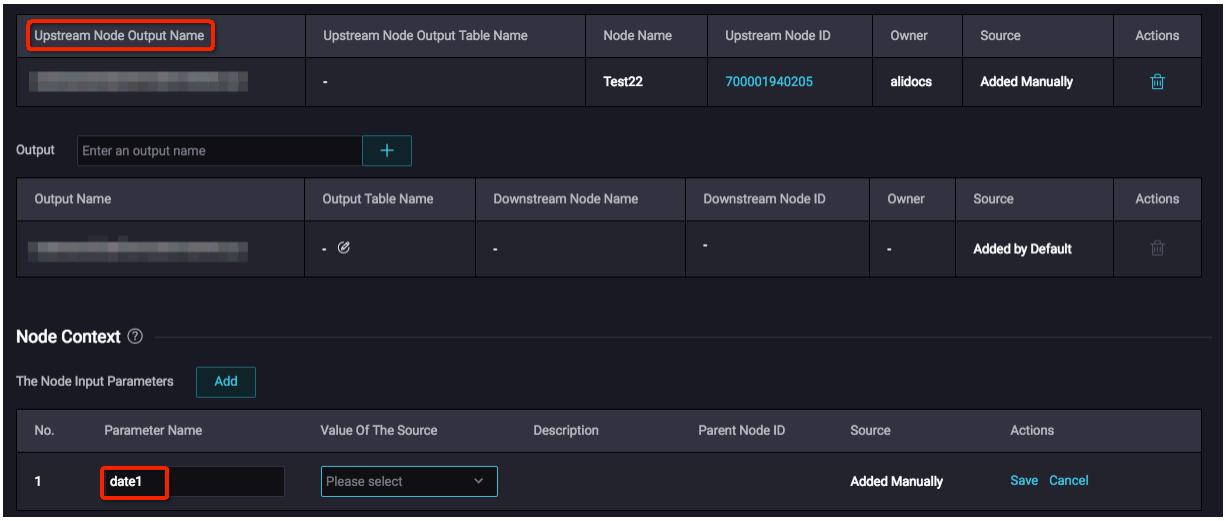
ノード **"test22"** は、ノード **"test223"** の上位ノードです。 ノード ” **test22**” で **[Node Context]** > **[The Node Output Parameters]** を選択し、設定を行います。 たとえば次の図に示すとおり、パラメーター名を`date1` 、値を `${yyyymmdd}` にして、**[Run]** をクリックします。



ノード **"test22"** を正常に送信した後、下位ノード **"test223"** を設定します。

> 注：
> **"test223"** の **[Dependencies]** > **[Upstream Node Output Name]** が **"test22"** の**[Dependencies]** > **[Output Name]** と同じであることを確認します。

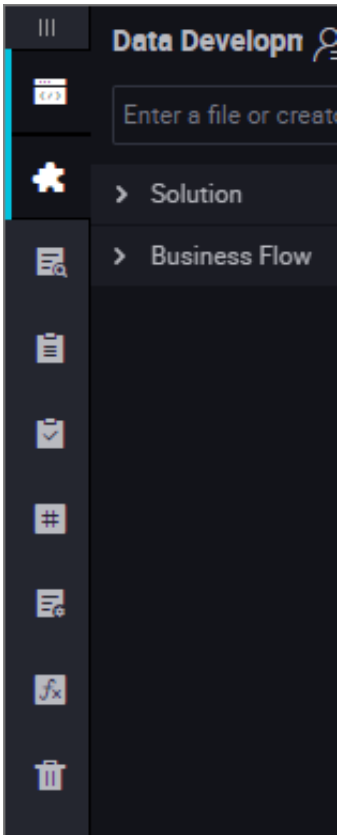
[Node Context] > **[The Node Input Parameter]** > **[Parameter Name]** の順に選択し、**"test22"** のパラメーター名 `"date1"` を 入力します。**[Value Of The Source]** のドロップダウンにオプションが表示されます。 該当するソースを選択し、**[Save]** を選択します。



1.6 設定管理

1.6.1 設定管理の概要

設定管理は、**DataStudio** インターフェイスの設定で、コード、フォルダー、テーマ、モジュールの追加や削除などを行います。 データ開発の左下隅にある歯車をクリックして設定管理ページへ移動します。

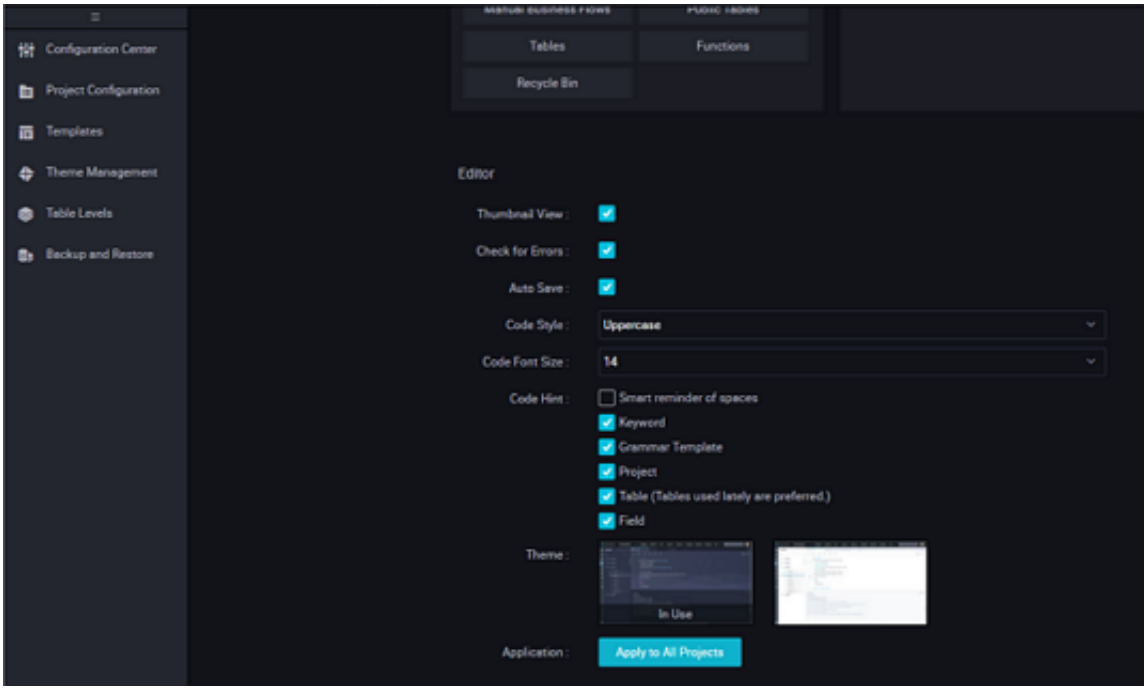


設定管理は、**5** つのモジュールに分かれています。 詳細は、以下のページをご参照ください。

- 設定センター
- プロジェクト設定
- テンプレート
- テーマの管理
- テーブルレベル

1.6.2 設定センター

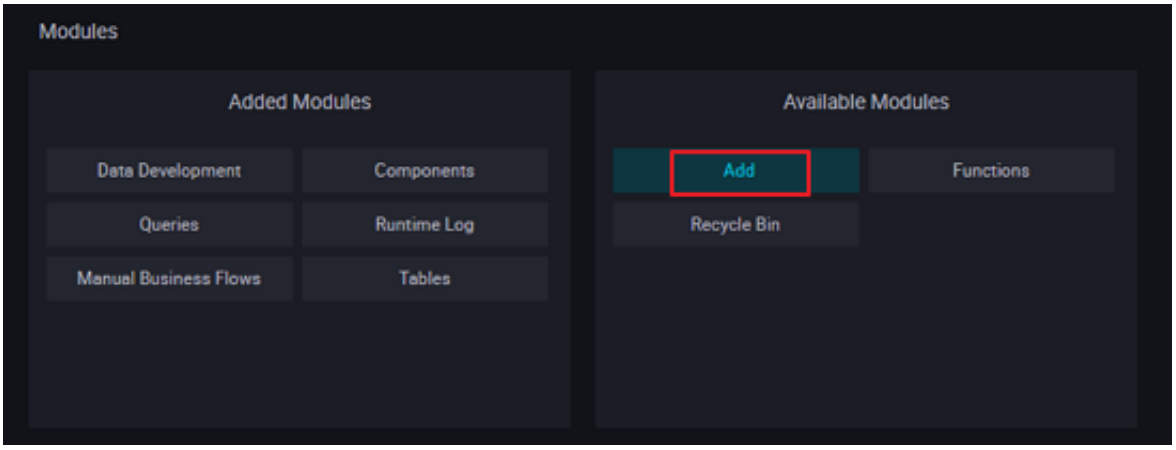
設定センターでは、モジュール管理とエディタ管理を含む共通機能の設定を行います。



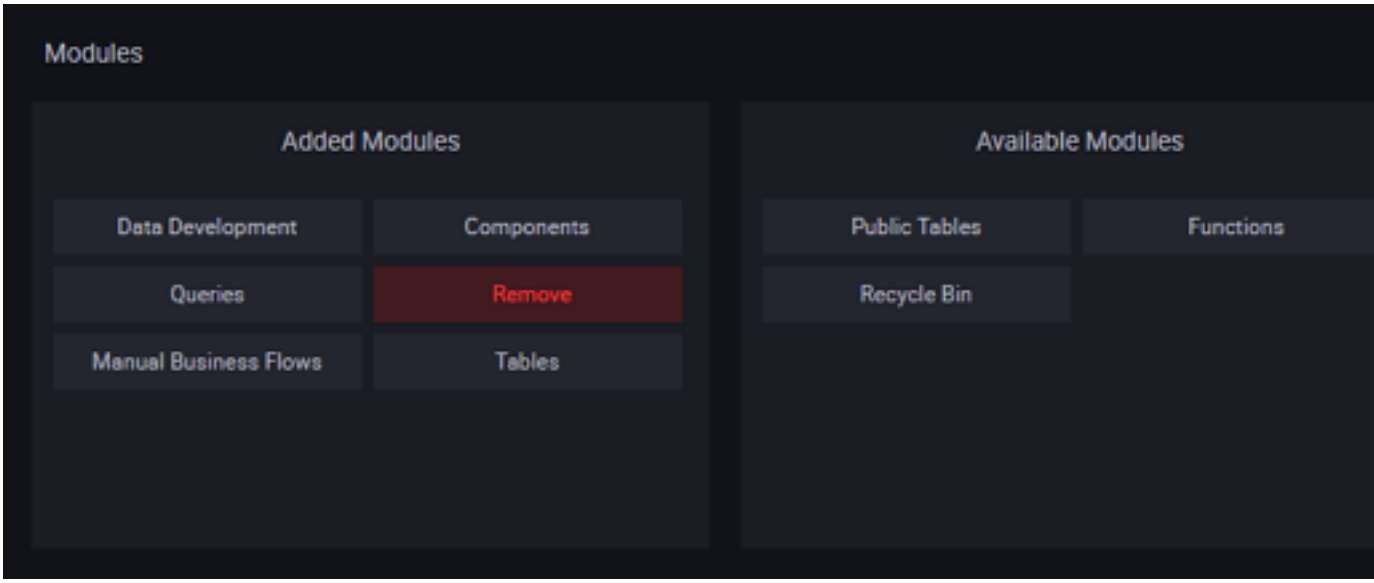
モジュール管理

モジュール管理では、**DataStudio** インターフェイスの左列の機能モジュールへモジュールを追加または削除する操作を行います。クリックして、左側に表示する必要のある機能モジュールにフィルターをかけたり、ドラッグ **&** ドロップでモジュールの機能をソートしたりできます。

追加するモジュールにマウスを置くと、モジュールは青色に変わり、**[Add]** が表示されます。



削除するモジュールにマウスを置くと、モジュールは赤色に変わり、**[Remove]** が表示されます。



注：

テンプレート管理フィルターはすぐに、現在のプロジェクトに対して有効になります。すべてのプロジェクトに対し有効にする場合は、**[the above settings to apply to all projects]** をクリックします。

エディタ管理

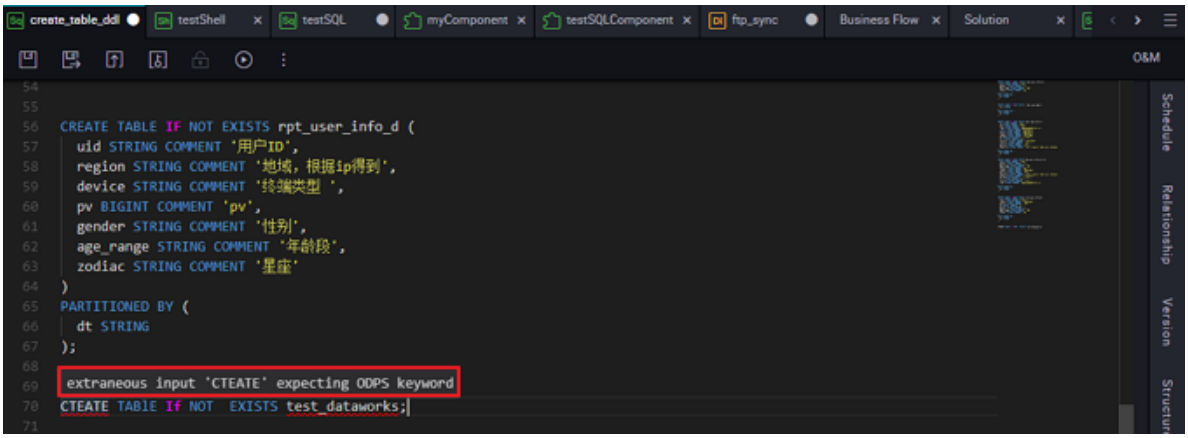
エディタではコードやキーワードの設定を行います。インターフェイスを更新することなく、リアルタイムで設定が有効になります。

- サムネイル表示

  現在のインターフェイスコードはコードの右側に表示されます。図の陰がついているエリアは、現在表示されているエリアを示します。コードが長い場合は、マウスを上下に移動させることで表示されるコードエリアを切り替えます。

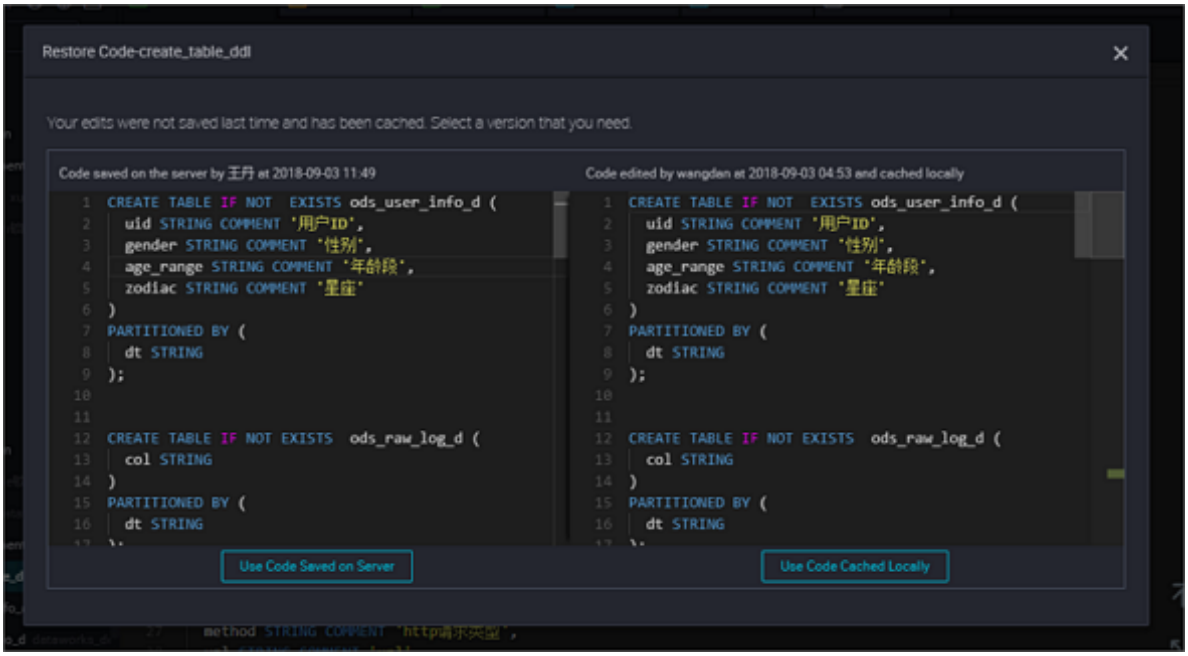
- エラーのチェック

  現在のコードのエラー状態を確認します。 赤いエラーコードエリアにマウスを置くと、エラー特有のフィールド状態が表示されます。



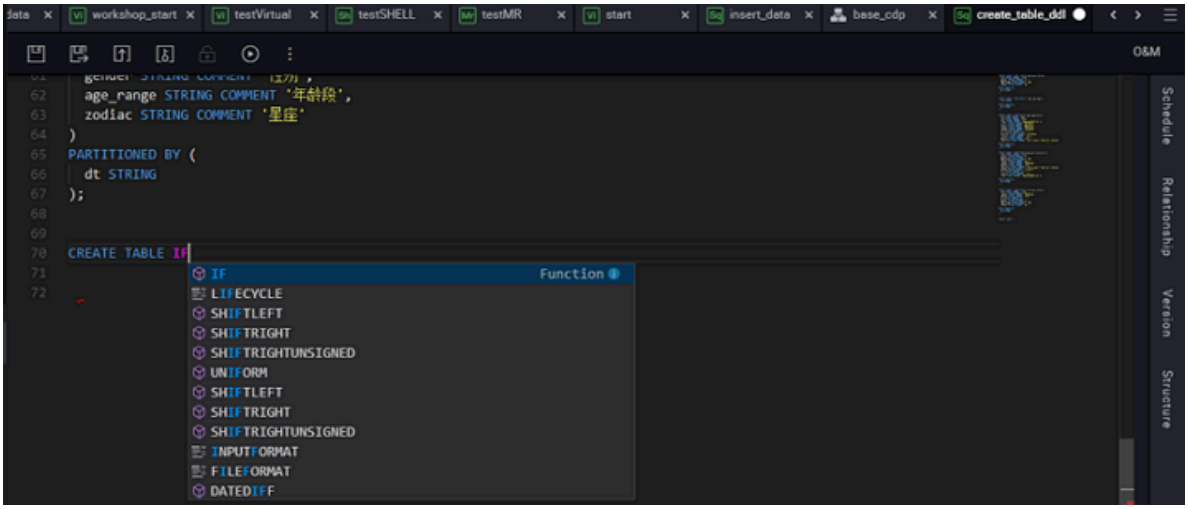
- 自動保存

  現在編集中のコードを自動的にキャッシュして、編集中のページのクラッシュやコードが保存されないことを防ぎます。 左側の **[Use server-saved code]**、または右側の **[Use locally cached code]** を選択します。



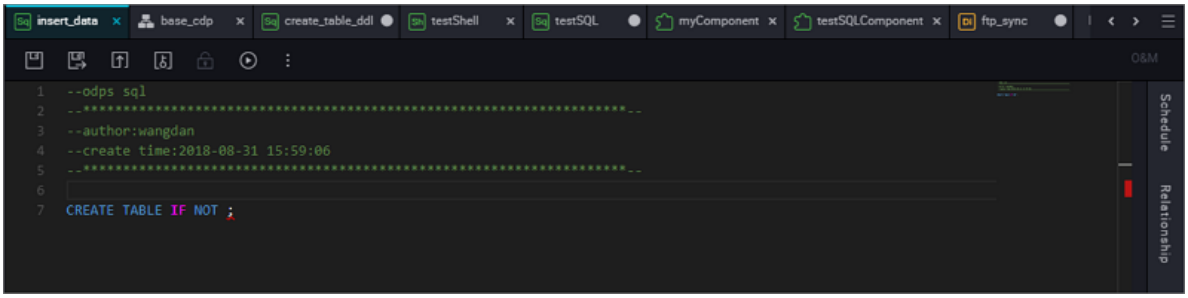
- コードスタイル

  コードスタイルは、お好みに応じて大文字、または小文字に設定することができます。 キーワードを入力し、**Enter** を押して **Lenovo** ショートカットを使って必要なキーワードを入力します。



- コードのフォントサイズ

  コードのフォントサイズは、最小 **12** 、最大 **18** フォントをサポートします。ご自身のコード書き込みの習慣や量に応じて、設定を変更します。



- コードのヒント

  コードのプロンプトはコードの入力の際に使用します。インテリジェントプロンプトの表示は、次のセクションに分かれています。

  - **Space Smart Tip: Lenovo** のキーワード、テーブル、フィールドを選択した後、スペースを追加します。
  - **keywords:** プロンプトコードは入力したキーワードをサポートします。
  - **Syntax template:** サポートされる構文テンプレートです。
  - **Project: Lenovo** のプロジェクト名を入力します。
  - **Table: Lenovo** で入力する必要のあるテーブルです。
  - **Field:** このテーブルのフィールドに対するスマートプロンプトです。

- テーマ

  テーマのスタイルは **DataStudio** インターフェイススタイルの設定です。現在は黒と白の両方をサポートしています。

- アプリケーション

  上記のテンプレート管理設定とエディタ管理設定を現在の既存プロジェクトに適用します。

1.6.3 プロジェクト設定

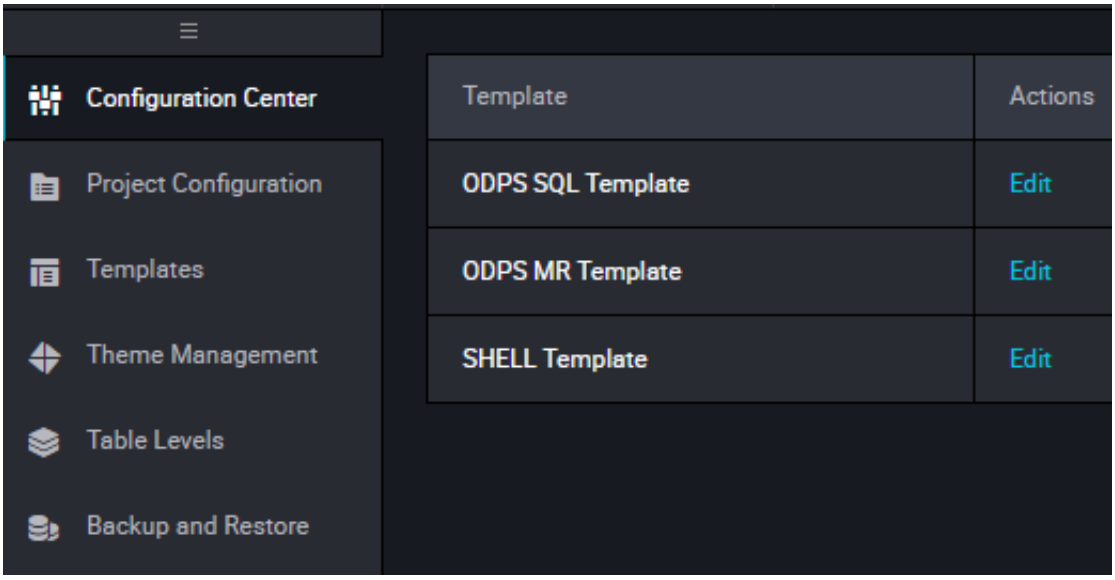
プロジェクト設定には、パーティション日形式、パーティションフィールド名、一時テーブルプレフィックス、アップロードテーブル (インポートテーブル) プレフィックスの **4** つの設定項目があります。

- **Partition Date Format:** デフォルトパラメーターです。コードのパラメーターの表示形式です。ご自身の要件に応じてパラメーターの形式を変更します。
- **Partition field naming:** パーティションデフォルトフィールド名
- **Temporary table prefix: "t_"** で始まるフィールドは、デフォルトでは一時テーブルとして認識されます。
- **Upload (import table) prefix: DataStudio** インターフェイスがテーブルをアップロードする際のテーブルの名前プレフィックスです。

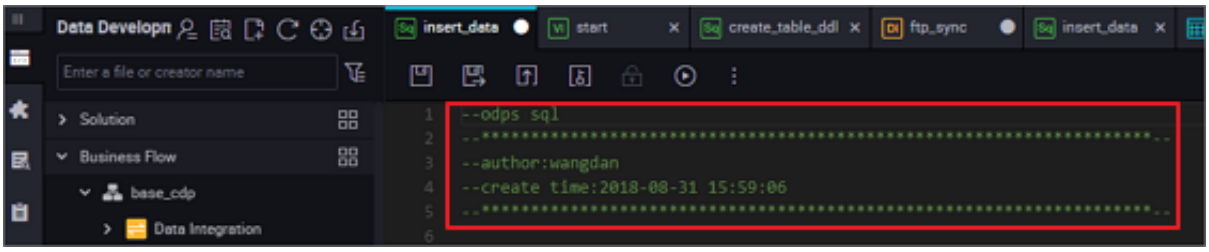
1.6.4 テンプレート

テンプレート管理はノードが作成された後、デフォルトでコードの前に表示されるコンテンツです。プロジェクト管理者は、必要に応じてテンプレートの表示スタイルを変更することができます。

現在、**ODPS SQL** テンプレート、**ODPS MR** テンプレート、**ODPS PL** テンプレート、**PERL** テンプレートおよび **SHELL** テンプレートに対してタイトルが設定されます。

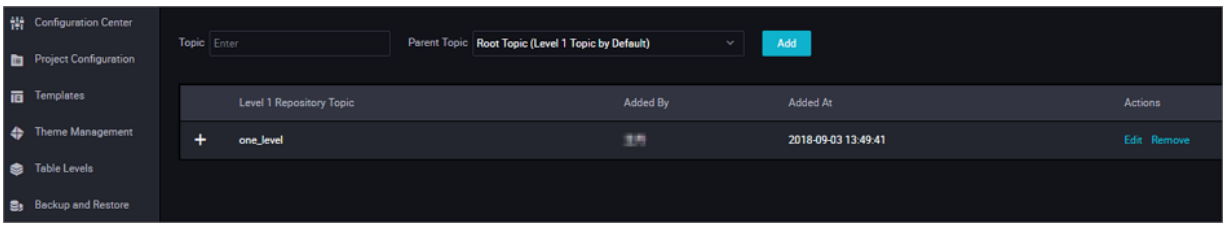


SQL ノードを例にとると、テンプレートの表示スタイルは次のとおりです。



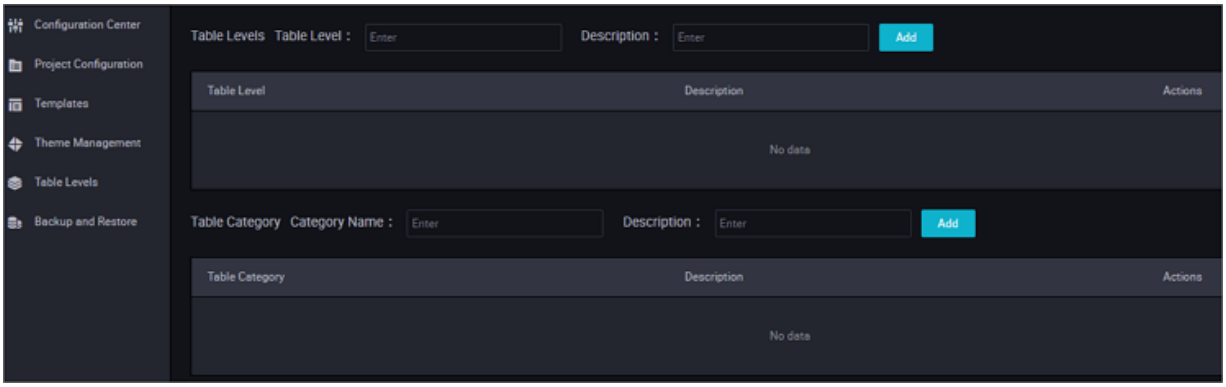
1.6.5 テーマの管理

テーブル管理にはたくさんのテーブルがあります。選択したトピックに応じて、テーブルは第二レベルのサブフォルダに格納されます。 これらのフォルダはテーブルにまとめられ、これがテーマです。 管理者は、プロジェクトの要件に応じて複数のテーマを追加したり、テーブルの目的や名前に応じて分類分けをしたり整理したりすることができます。



1.6.6 テーブルレベル

テーブルレベルは、テーブルの物理的なレベルデザインです。 問題が発生しプロジェクトの影響が正確に特定できず、それがオンライン操作の通常の操作へつながる場合、プロジェクトに対するテーブルの重要性に応じて、問題が影響することを防ぐためにテーブルが分割されます。

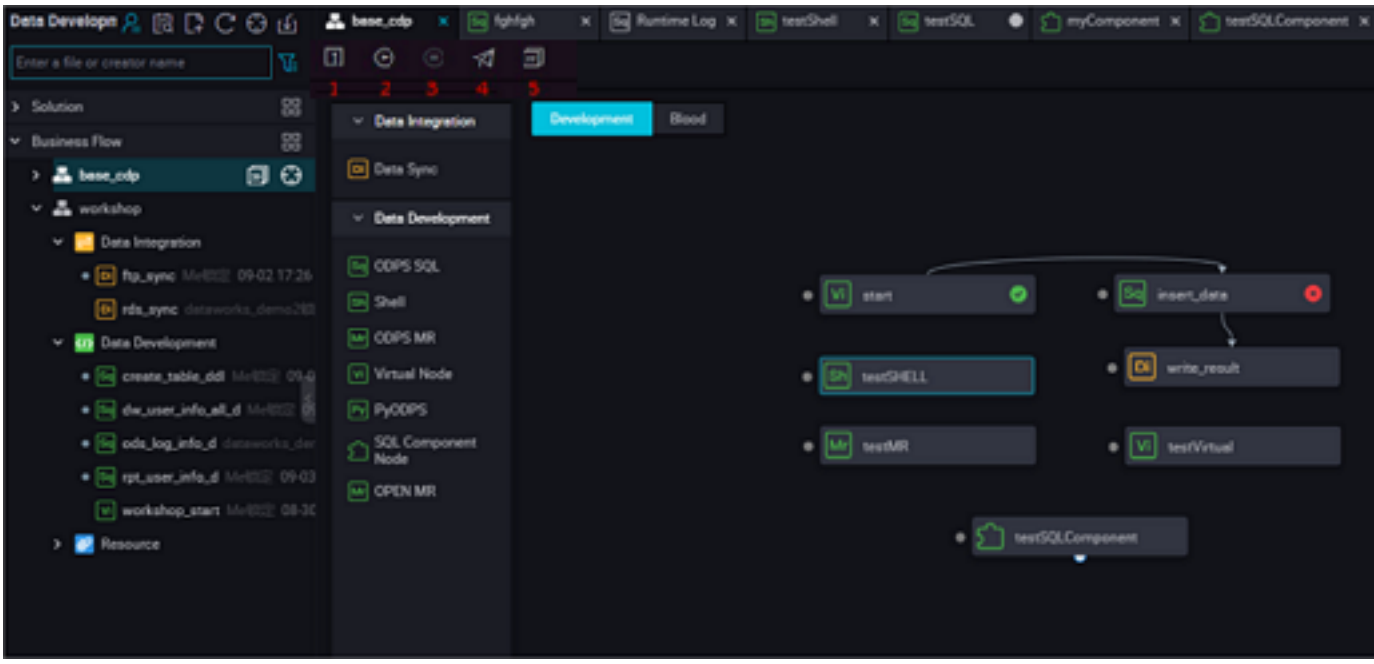


プロジェクトにデフォルトの階層はありません。管理者はプロジェクトの目的や必要性に応じて手動で追加する必要があります。

1.7 マニュアルビジネスフロー

1.7.1 マニュアルビジネスフローの紹介

マニュアルビジネスフロー (**Manual Business Flow**) では、作成されたすべてのノードを手動でトリガーする必要があり、スケジューリングを使って実行することができません。 したがって、マニュアルビジネスフローでは、親ノードの依存関係とノードのローカルノード出力を設定する必要がありません。



マニュアルビジネスフローインターフェイスの機能は次のとおりです。

No.	機能	説明
1	**Submit**	クリックして、現在のマニュアルビジネスフローにあるすべてのノードを送信します。

No.	機能	説明
2	**Run**	クリックして、現在のマニュアルビジネスフローにあるすべてのノードを実行しまうｓ。 手動タスクに依存関係がない場合、これらのタスクは同時に実行されます。
3	**Stop Run**	クリックして、実行中のノードを停止します。
4	**Publish**	クリックしてタスク発行インターフェイスへ移動します。タスク発行インターフェイスでは、すでに送信されているがまだ運用環境へ発行されていないノードの一部またはすべてを発行することができます。
5	**Go to O&M**	クリックして **O&M Center** へ移動します。
6	**Reload**	クリックして、現在のマニュアルビジネスフローインターフェイスを再度読み込みます。
7	**Auto Layout**	クリックして、現在のマニュアルビジネスフローインターフェイスにあるノードを自動的にシーケンスします。
8	**Zoom-in**	クリックしてインターフェイスをズームインします。
9	**Zoom-out**	クリックしてインターフェイスをズームアウトします。
10	**Query**	クリックして、現在のマニュアルビジネスフローにあるノードを照会します。
11	**Full Screen**	クリックして、現在のマニュアルビジネスフローにあるノードを全画面表示モードで表示します。
12	**Parameters**	クリックして、パラメーターを設定します。 フローパラメーターの優先順位は、ノードパラメーターより高くなります。 パラメーターキーがパラメーターと一致する場合、ビジネスフローパラメーターは優先的に設定されます。
13	**Operation Records**	クリックして、現在のマニュアルビジネスフローにあるすべてのノードの操作履歴を表示します。
14	**Version**	クリックして、現在のマニュアルビジネスフローにあるすべてのノードの送信履歴と発行履歴を表示します。

1.7.2 リソース

リソース **(Resource)** は **ODPS** 特有のコンセプトです。 リソースは **ODPS UDF** または **ODPS MR** を使用する際に利用できます。

- **ODPS SQL UDF: UDF** のコンパイル後、コンパイルされた **jar** パッケージを **ODPS** をアップロードする必要があります。 この **UDF** を実行する際、**ODPS** は自動的に **jar** パッケージをダ

ウンロードし、ユーザーコードを抽出し、**UDF** を実行します。 **jar** パッケージのアップロードプロセスは、**ODPS** でリソースを作成するプロセスです。 **jar** パッケージは **ODPS** リソースのタイプの **1** つです。

- **ODPS MapReduce: MapReduce** プログラムをコンパイル後、コンパイルされた **jar** パッケージをリソースとして **ODPS** へアップロードする必要があります。 **MapReduce** ジョブを実行する際、**MapReduce** フレームワークはこの **jar** パッケージを自動的にダウンロードし、ユーザーコードを抽出します。

同様に、テキストファイル、**ODPS** テーブル、およびさまざまな圧縮パッケージ **(.zip**、**.tgz** 、**.tar.gz**、**.tar** および **.jar** など) をアップロードすることができます。 これで、**UDF** や **MapReduce** を実行する際リソースを読み込んだり使用したりできるようになります。

ODPS では、リソースの読み込みや使用のために **API** が提供されています。 **ODPS** リソースの次のタイプが利用可能です。

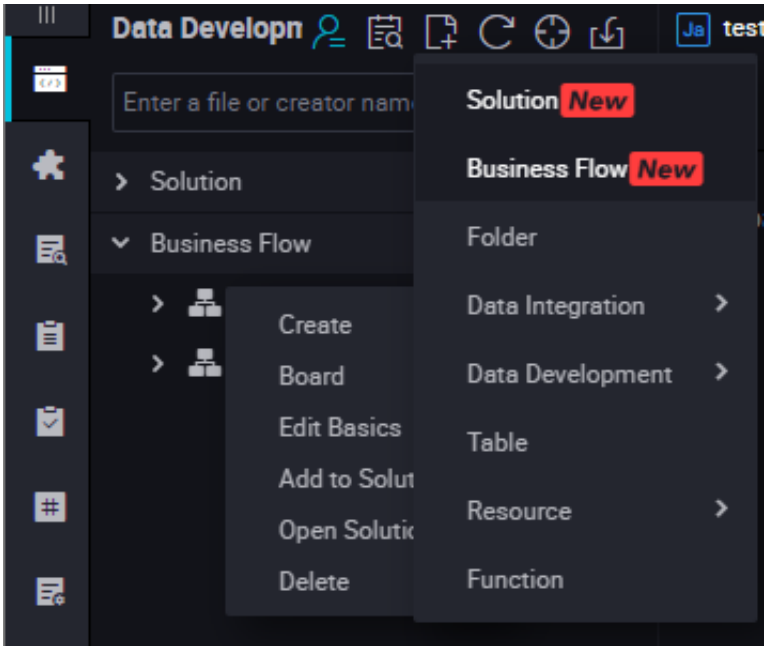
- ファイル
- アーカイブ**:** リソース名の拡張子で圧縮タイプを識別します。 次の圧縮ファイルタイプがサポートされています。**.zip**、**.tgz**、**.tar.gz**、**.tar** および **.jar** です。
- **Jar:** コンパイルされた **Java jar** パッケージです。

DataWorks では、リソースの作成プロセスはリソースの追加プロセスと同じです。 現在、**DataWorks** では、**3** つのタイプのリソースの視覚的な追加をサポートしています。**jar**、**Python**、ファイルリソースです。 新しく作成されたエントリは同じです。違いは次のとおりです。

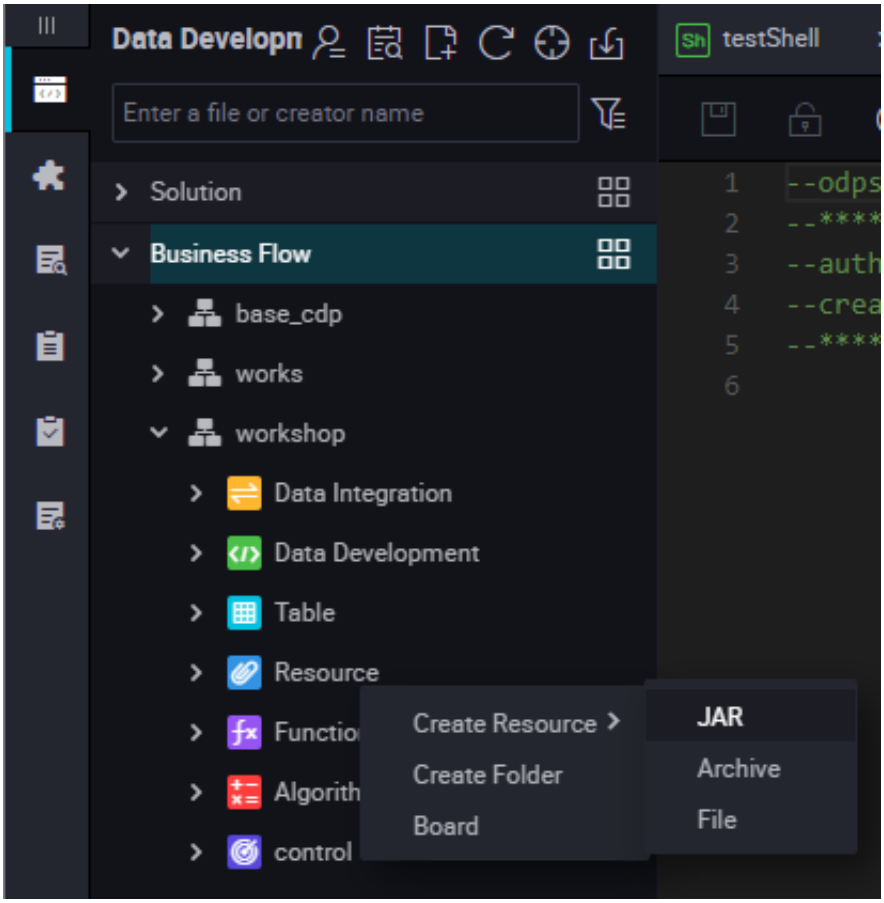
- **Jar** リソース**: Java** コードをオフライン **Java** 環境でコンパイルし、コードを **jar** パッケージへ圧縮し、**ODPS** へ **jar** リソースとしてパッケージをアップロードする必要があります。
- スモールファイル**:** これらのリソースは **DataWorks** で直接編集されます。
- ファイルリソース**:** ファイルリソースを作成する際、ビッグファイルを選択する必要があります。 ローカルリソースファイルもアップロードできます。

リソースインスタンスの作成

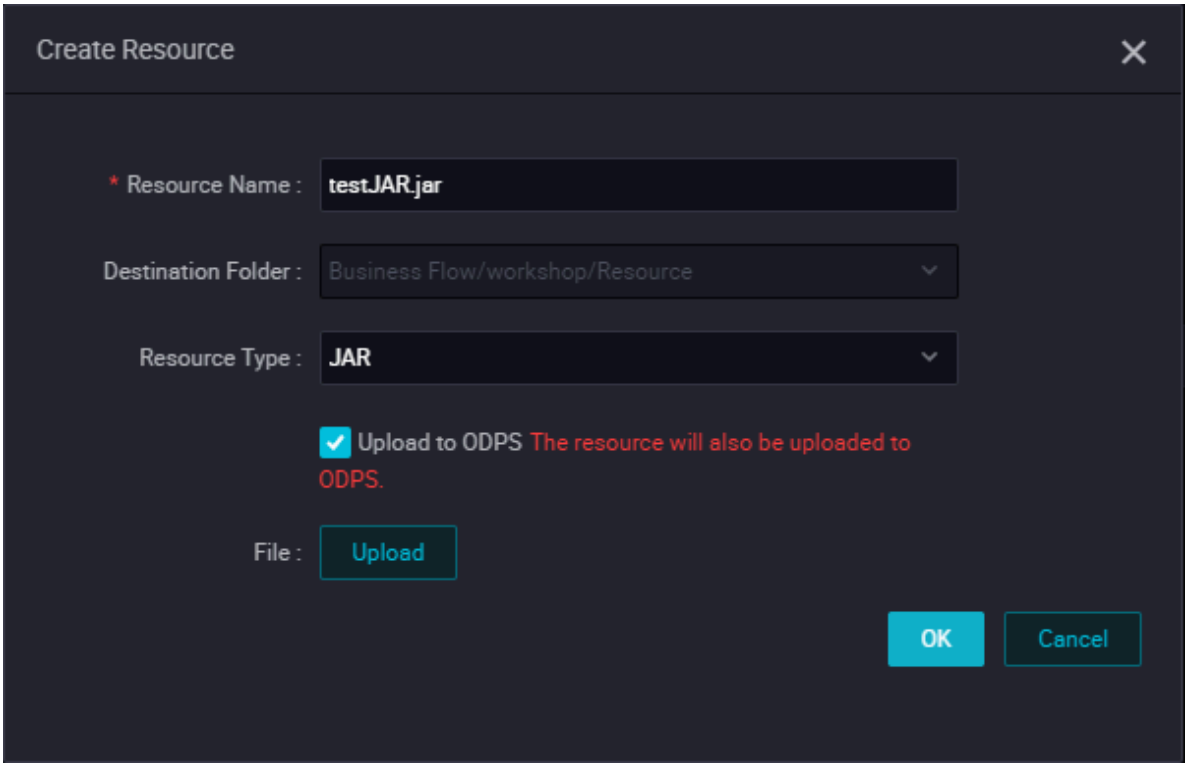
1. 左側のナビゲーションバーで **[Manual Business Flow]** をクリックし、**[Create Business Flow]** を選択します。



2. **[Resource]** を右クリックし、**[Create Resource]** > **[jar]** を選択します。



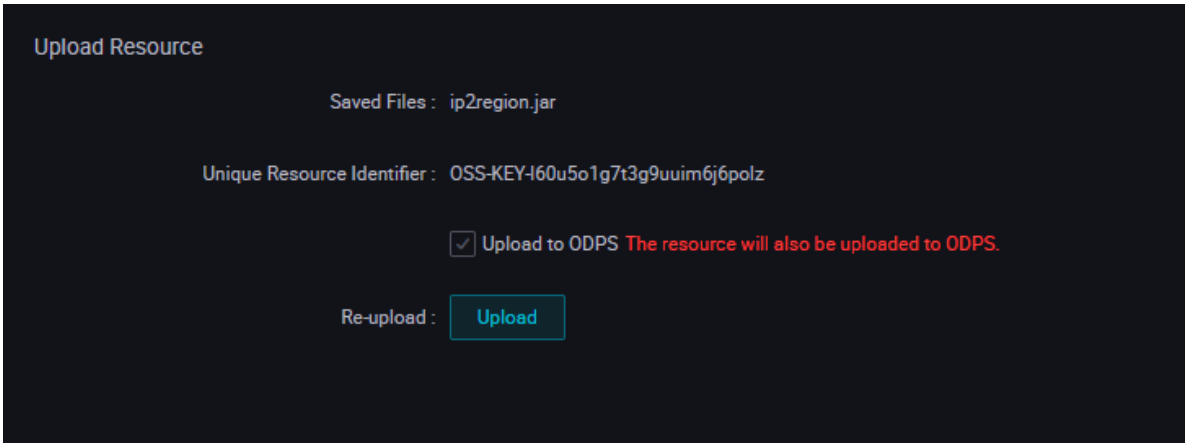
3. **[Create Resource]** ダイアログボックスが表示されます。 名前付け規則に沿って、リソース名を入力し、リソースタイプを **jar** に設定します。アップロードするローカル **jar** パッケージを選択します。 **[Submit]** をクリックし、開発環境へパッケージを送信します。



注 :

- **ODPS** クライアントへこの **jar** パッケージがアップロードされた後、**[Uploaded as the ODPS resource]** の選択を解除する必要があります。 このアップロードでは、リソースも **ODPS** へアップロードされます。 そうでない場合、アップロードプロセス中にエラーが報告されます。
- リソース名はアップロードするファイル名と同じである必要はありません。
- リソース名の名前付け規則**: 1** から **128** 文字以内の文字列で、文字、数字、アンダースコア、ドットを含みます。 名前は大文字と小文字が区別されます。 リソースが **jar** リソースである場合、拡張子は **.jar** です。

4. **[Submit]** をクリックし、リソースを開発スケジューリングサーバーへ送信します。



5. ノードタスクを解放します。

   操作に関する詳細情報は、「*#unique_40*」を参照してください。

1.7.3 関数

UDF の登録

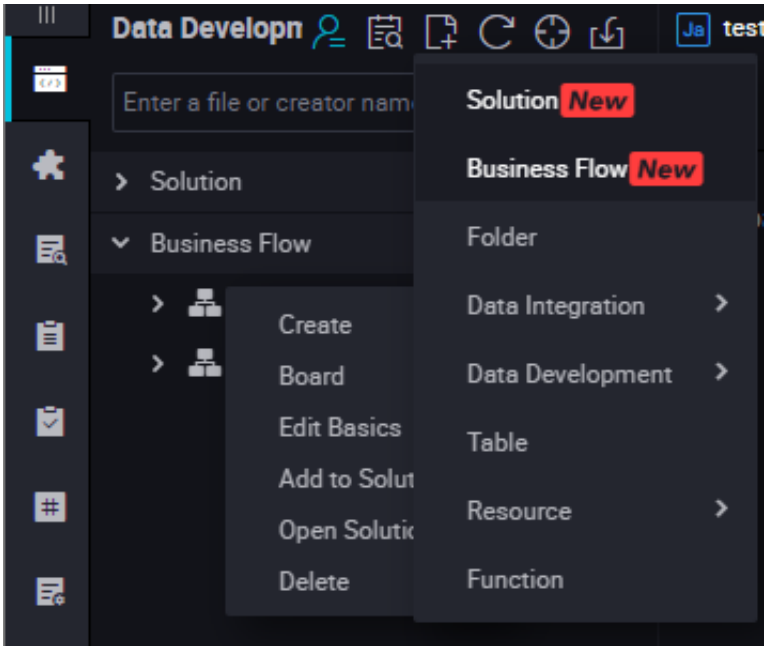
MaxCompute では **UDF** をサポートしています。 詳細については、「*UDF の概要*」をご参照ください。

DataWorks では、`add function` という **ODPS** コマンドラインを置換するために関数を登録する視覚的な **GUI** が提供されています。

現在、**Python** と **Java API** が **UDF** の実装をサポートしています。 **UDF** プログラムをコンパイルするには、「リソースの追加」 を参考に **UDF** をアップロードし、**UDF** を登録します。

UDF の登録手順

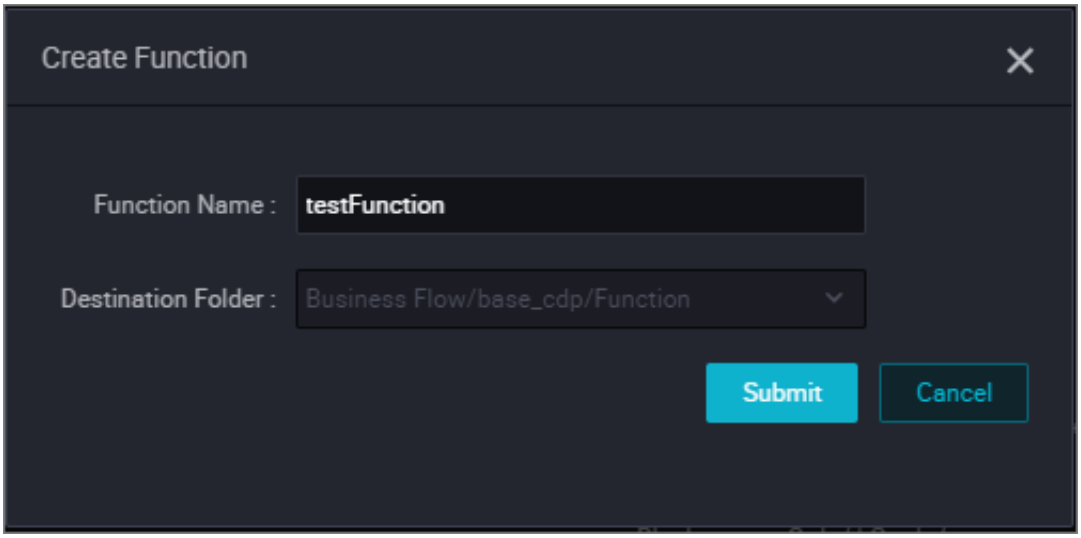
1. 左側のナビゲーションウィンドウで **[Manual Business Flow]** をクリックし、**[Create Business Flow]** を選択します。



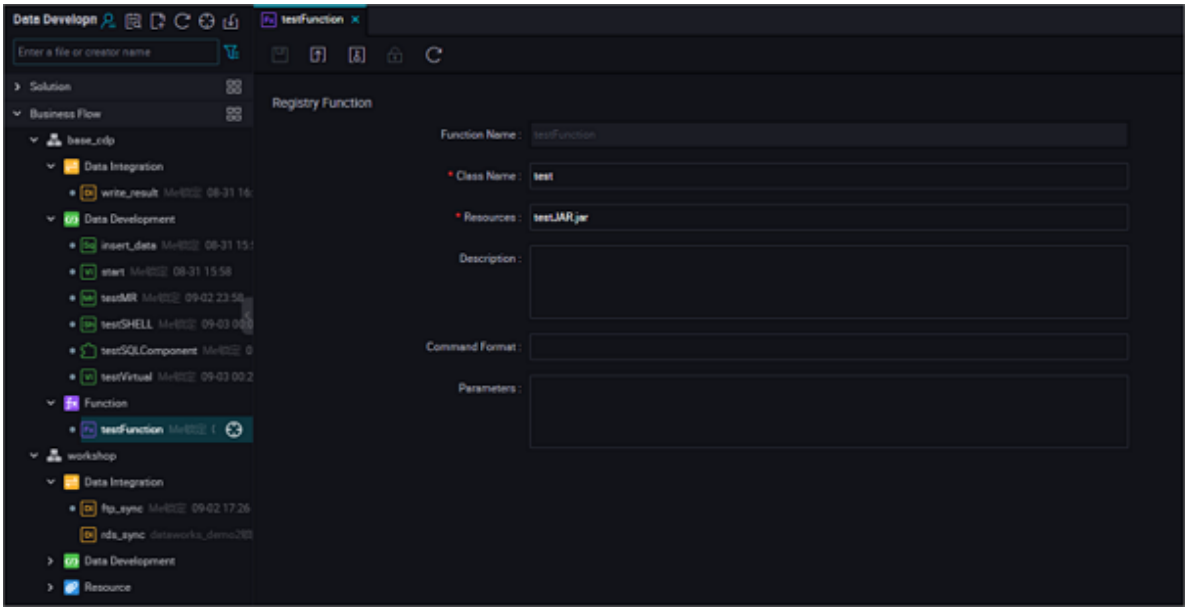
2. オフライン **Java** 環境で、プログラムを編集し、プログラムを **jar** パッケージへ圧縮し、**jar** リソースを作成します。そしてプログラムを送信し解放します。

   別の方法としては、**Python** リソースを作成し、**Python** コードをコンパイルし保存します。そしてコードを送信し解放します。 詳細については、「リソースの作成」 をご参照ください。

3. **[Function]** > **[Create Function]** を選択し、新しい関数の設定を入力し、**[Submit]** をクリックします。



4. 関数の設定を編集します。



- **Class Name: UDF** を実装するメインクラスの名前です。 リソースが **Python** の場合、典型的な書き込みスタイルは、**" Python** リソース名**.**クラス名**"** (リソース名に **py** は不要です)。
- **Resources:** 第二段階のリソース名です。複数のリソースがある場合は、コンマで区切ります。
- **Description: UDF** の説明です。 オプションです。

5. ジョブを送信します。

   設定が完了したら、ページの左上隅にある**[Save]** をクリックする、または **ctrl** + **S** を押してノードを開発環境へ送信 (ロックの解除) します。

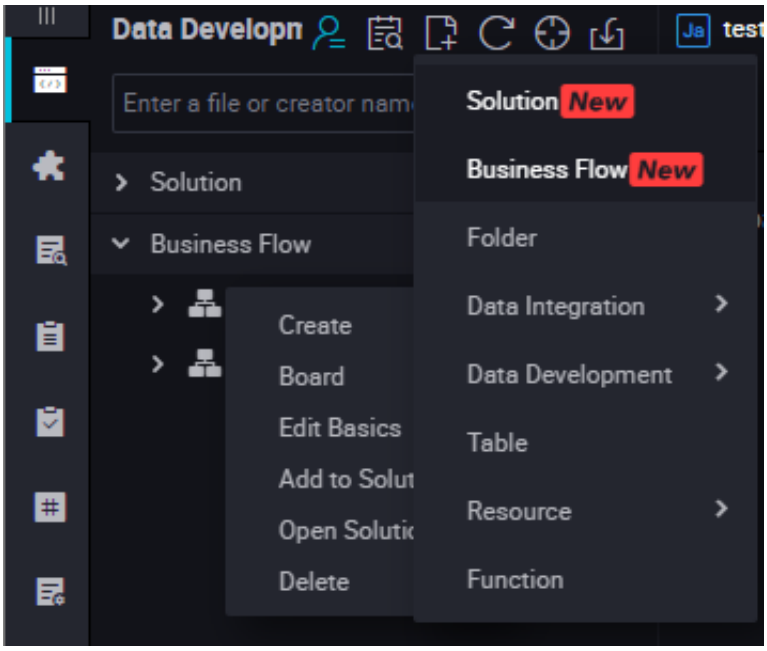
6. ノードタスクを解放します。

   操作に関する詳細は、「*#unique_40*」をご参照ください。

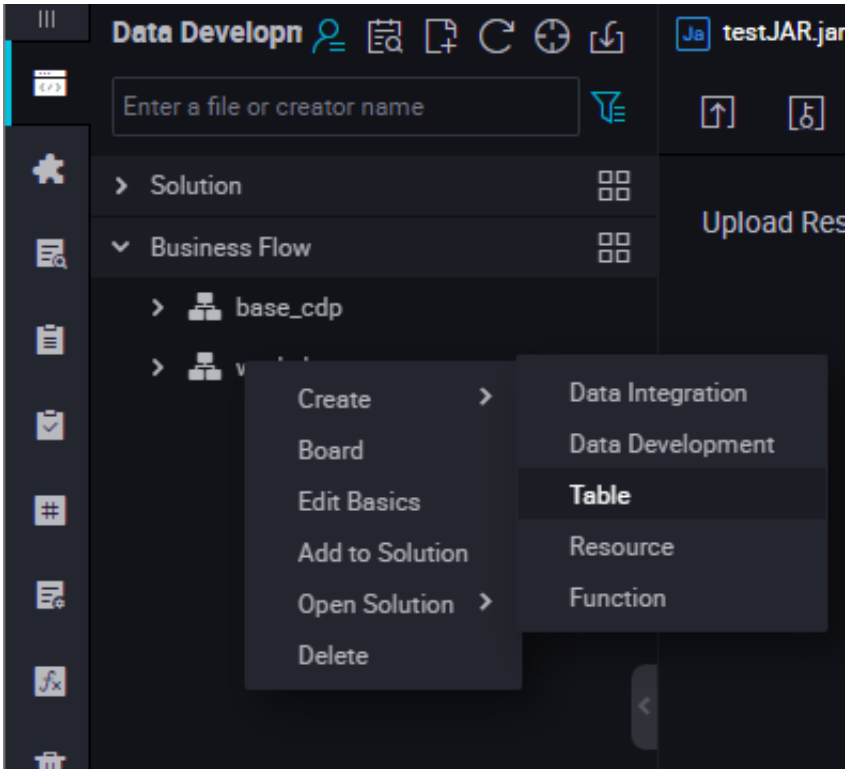
1.7.4 テーブル

テーブルの作成

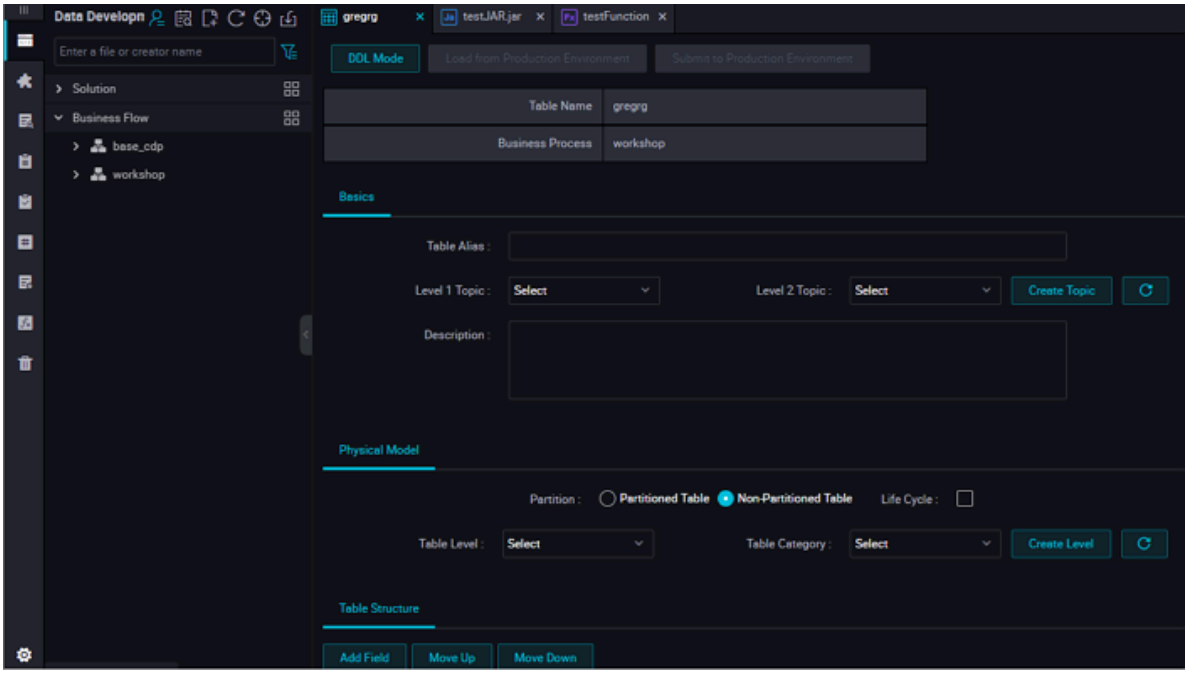
1. **[Manual Business Flow]**、**[Create Business Flow]** の順に選択します。



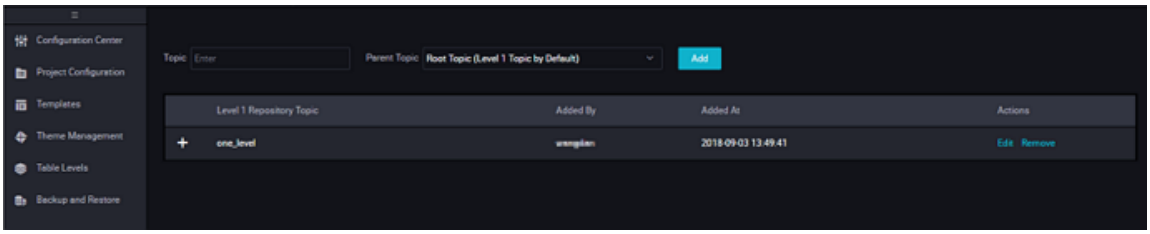
2. **[Table]** を右クリックし、**[Create Table]** を選択します。



3. 基本属性を設定します。

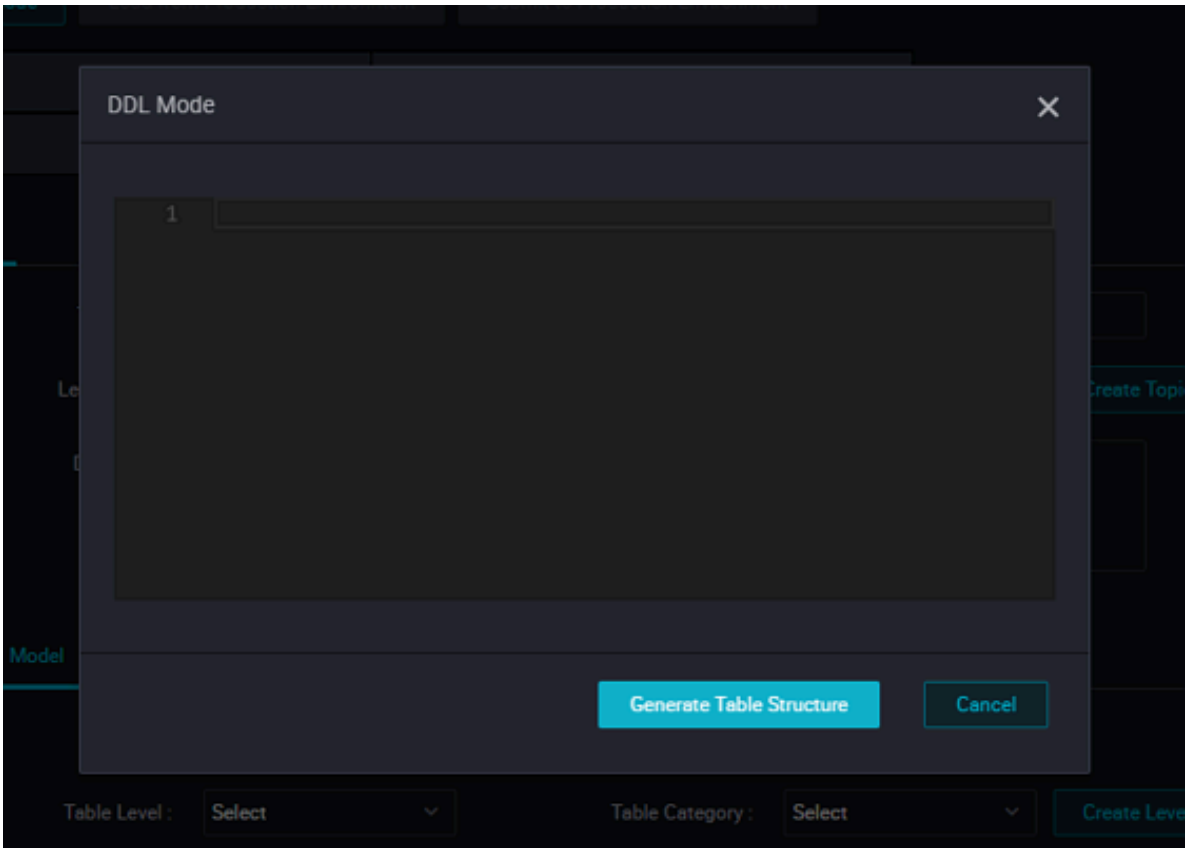


- **Chinese Name:** 作成するテーブルの中国語名です。
- **Level-1 Topic:** 作成するテーブルの 第一階層のフォルダ名です。
- **Level-2 Topic:** 作成するテーブルの第二階層フォルダ名です。
- **Description:** 作成するテーブルの説明です。
- **[Create Topic]** をクリックします。 表示された **[Topic Management]** ページで、第 **1** 階層と第 **2** 階層を作成します。



4. **DDL** モードでテーブルを作成します。

   [DDL Mode] をクリックします。 表示されたダイアログボックスで、標準テーブルの作成文を入力します。



   テーブル作成の **SQL** 文を編集した後、**[Generate Table Structure]** をクリックします。 **[Basic Attributes]** 領域、**[Physical Model Design]** 領域、**[Table Structure Design]** 領域内の情報が自動的に入力されます。

5. **GUI** でテーブルを作成します。

   DDL モードでのテーブル作成が難しい場合には、次の設定を行うと、**GUI** でテーブルを作成できます。

   - **[Physical model design]** 領域
     - **Partition Type:**「**Partitioned Table**」または「**Non-partitioned Table**」を設定できます。
     - **Life Cycle: MaxCompute** のライフサイクル機能です。 **Life Cycle** (単位: 日) で指定した期間内にアップロードされていないテーブル (またはパーティション) 内のデータは消去されます。
     - **Level:**「**DW**」、「**ODS**」、「**RPT**」のいずれかを設定できます。
     - **Physical Category:**「**Basic Business Layer**」、「**Advanced Business Layer**」、「**Other**」のいずれかを設定できます。 **[Create Level]** をクリックします。表示された **[Level Management]** ページで階層を作成します。
   - **[Table structure design]** 領域
     - **English Field Name:** フィールドの英語名です。文字、数字、アンダースコア(_) を使用できます。
     - **Chinese Name:** フィールドの略称された中国名
     - **Field Type: MaxCompute** のデータタ型です。「**String**」、「**Bigint**」、「**Double**」、「**Datetime**」、「**Boolean**」のみ使用できます。
     - **Description:** フィールドの詳しい説明です。
     - **Primary Key:** フィールドが主キーか、結合条件の主キーの場合に選択します。
     - 新しいフィールドに列を追加するには、**[Add Field]** をクリックします。
     - 作成したフィールドを削除するには、**[Delete Field]** をクリックします。

        注 :
       作成したテーブルからフィールドを削除した後、そのテーブルを再度送信する場合、現行テーブルを破棄して、同じ名前で新しいテーブルを作成する必要があります。 この操作は、本番環境で行うことはできません。
     - 作成するテーブルのフィールドの順番を調整するには、**[Move Up]** をクリックします。ただし、作成したテーブルのフィールドの順番を調整する場合、現行テーブルを破棄し

て、同じ名前で新しいテーブルを作成する必要があります。 この操作は、本番環境で行うことはできません。

- **[Move Down]** の操作は、**[Move Up]** の場合と同じです。
- 現行テーブルにパーティションを作成するには、**[Add Partition]** をクリックします。作成したテーブルにパーティションを追加する場合、現行テーブルを破棄して、同じ名前で新しいテーブルを作成する必要があります。 この操作は、本番環境で行うことはできません。
- パーティションを削除するには、**[Delete Partition]** をクリックします。 作成したテーブルからパーティションを削除する場合、現行テーブルを破棄して、同じ名前で新しいテーブルを作成する必要があります。 この操作は、本番環境で行うことはできません。
- **Action:** 新しいフィールドの追加、フィールドの削除、および他の属性の編集を確認します。

  他の属性には、データ品質に関連する情報が含まれます。これはシステムに対して、検証ロジックを生成するために提供されています。 データプロファイル、**SQL** スキャン、テストルールの生成などのシナリオで使用されます。

  ■ **0 Allowed:** 選択するとフィールド値をゼロにすることができます。 このオプションは、**Bigint** 型と **Double** 型のフィールドにのみ適用されます。

  ■ **Negative Value Allowed:** 選択すると、フィールド値を負の数にすることができます。 このオプションは、**Bigint** 型と **Double** 型のフィールドにのみ適用されます。

  ■ **Security Level:**「**[Non-sensitive**」、「**Sensitive**」、「**Confidential**」のいずれかに設定できます。

  ```
  C: 顧客データ、B: 企業データ、S: ビジネスデータ
  C1-C2、B1、S1 は 非機密 (non-sensitive) データです。
  C3、B2-B4、S2、S3 は機密 (sensitive) データです。
  C4、S4、B4 は極秘 (Confidential) データです。
  ```

  ■ **Unit:** 金額の単位で、ドルやセントを選択できます。 このオプションは金額に関連のないフィールドには必要ありません。

  ■ **Lookup Table Name/Key Value:** 会員タイプやステータスといった列挙型フィールドに適用されます。 このフィールドに対応した辞書テーブル (またはディメンションテーブル) の名前を入力します。 たとえば、会員ステータスに対応した辞書テーブル名は、**"dim_user_status"** です。 グローバルユニークな辞書テーブルを使用する

場合、辞書テーブル内のフィールドに対応する **key_type** を入力します。 たとえば、会員ステータスの対応キー値は、**"TAOBAO_USER_STATUS"** です。

- **Value Range:** 現行フィールドの最大値と最小値です。 **Bigint** 型 と **Double** 型のフィールドにのみに適用されます。
- **Regular Expression Verification:** 現行フィールドで使用する正規表現です。 たとえば、フィールドが携帯電話番号で、正規表現 (またはさらに厳格な制限) で **11** 桁の数字に制限することができます。
- **Maximum Length:** フィールド値の最大文字数です。 **String** 型のフィールドにのみ適用されます。
- **Date Precision:** 日付の精度です。「**Hour**」、「**Day**」、「**Month**」、「**others**」のいずれかに設定できます。 たとえば、フィールド値が **2014-08-01**であっても (精度は「**Day**」のように見えます)、月次サマリーテーブル **"month_id"** の精度は「**Month**」です。 **Datetime** 型 または **String** 型の日付値に適用されます。
- **Date Format: String** 型の日付値にのみ適用されます。 フィールドに実際に格納される日付値の形式は、**yyyy-mm-dd hh:mm:ss** のようになります。
- **KV Primary Separator/Secondary Separator:** キー値ペアが組み合わされた大規模なフィールド **(String** 型) に適用されます。 たとえば、プロダクト拡張属性に **"key1:value1;key2:value2;key3:value3;..."** といった値がある場合、セミコロン **(;)** はフィールドの主となるセパレーターで、キー値ペアを区切ります。コロン **(:)** は二次的なセパレーターで、キー値ペアのキーと値を区切ります。

- **Partition Field Design:** このオプションは、**[Physical Model Desing]** 領域の **[Partition Type]** が「**Partitioned Table**」に設定されている場合にのみ表示されます。
- **Field Type:** すべてのフィールドに対して、**String** 型を使用することを推奨します。
- **Date Partition Format:** パーティションフィールドが日付 (データ型は **String)** の場合、「**yyyymmmdd**」といった日付形式を選択または入力します。
- **Date Partition Granularity:** たとえば、「**Day**」、「**Month**」、「**Hour**」 です。

テーブルを送信します。

テーブル構造情報を編集した後、新規テーブルを開発環境および本番環境へ送信します。

- **[Load from Development Environment]** をクリックします。 テーブルが開発環境に送信されると、このボタンはハイライトされます。 このボタンをクリックすると、開発環境で作成したテーブルの情報が、現在のページ情報に上書きされます。
- **[Submit to Development Environment]** をクリックすると、現在の編集ページで必要な項目がすべて完全に設定されているかどうか確認されます。 未設定の項目がある場合、テーブルの送信を禁止するアラームが報告されます。

- **[Load from Production Environment]** をクリックすると、本番環境へ送信したテーブルの詳細情報が、現在のページ情報に上書きされます。
- **[Create in Production Environment]** をクリックすると、本番環境のプロジェクトにテーブルが作成されます。

1.8 マニュアルタスクのノードタイプ

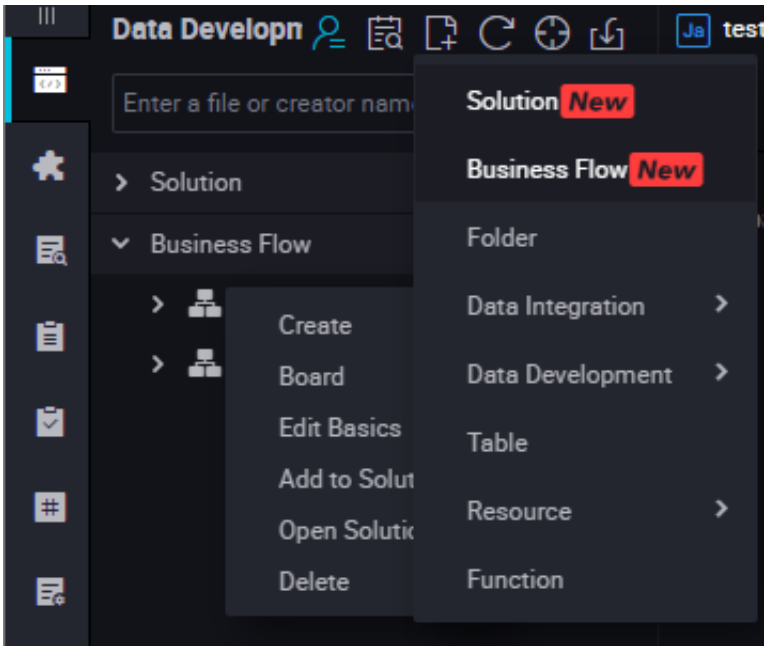
1.8.1 仮想ノード

仮想ノードは、データを生成しない制御ノードです。 通常、ワークフローのノード計画全体に対するルートノードとして使用します。

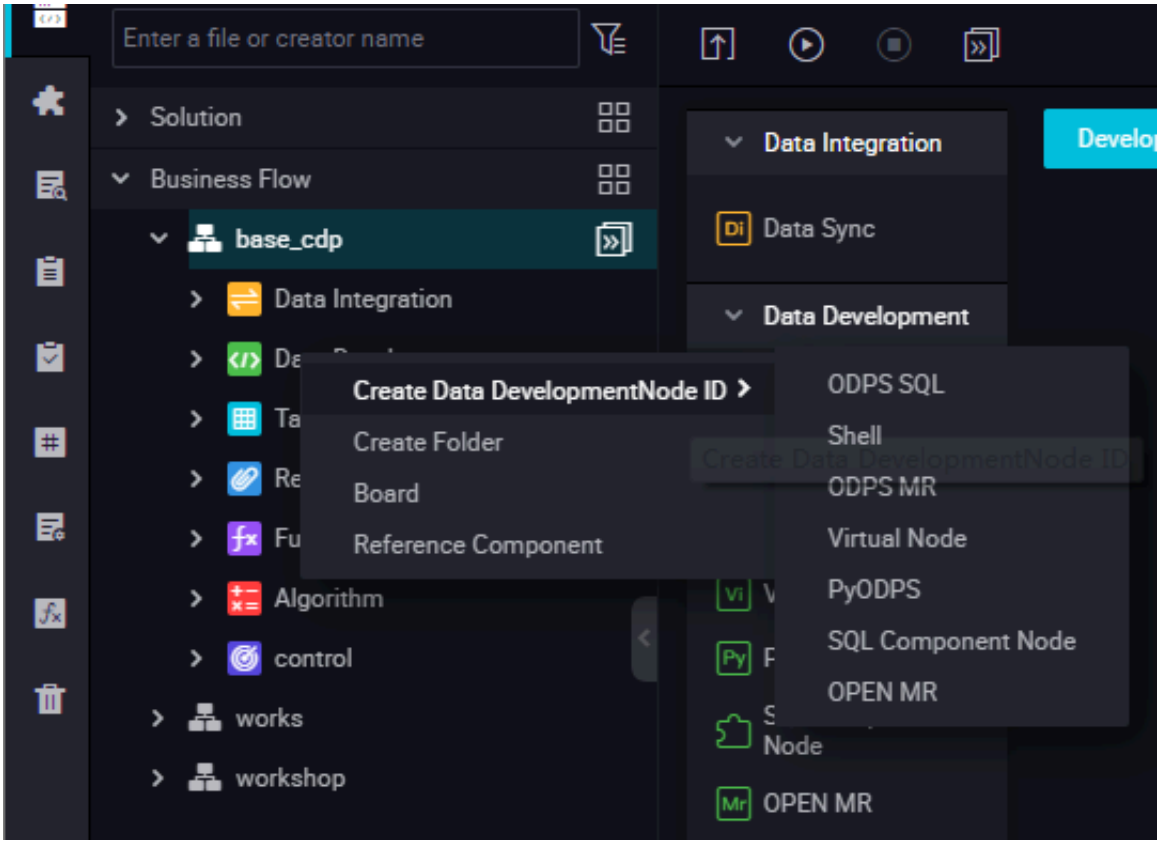
仮想ノードタスクの作成

1. ビジネスフローを作成します。

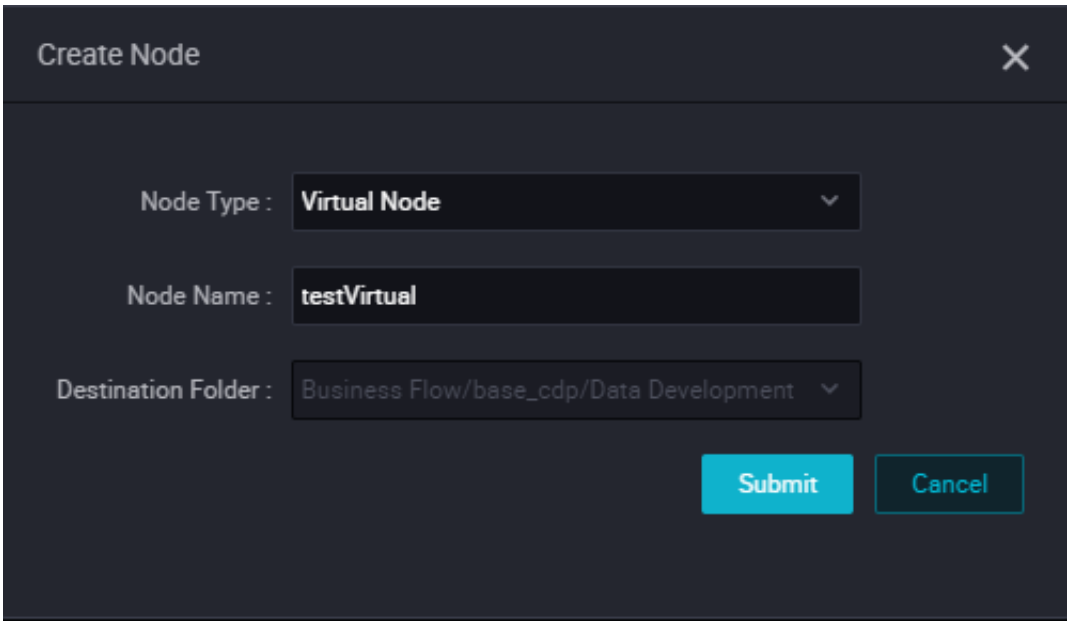
   左側のナビゲーションウィンドウで **[Manual Business Flow]** 、**[Create Business Flow]** の順に選択します。



2. 仮想ノードを作成します。**[Data Development]** を右クリックし、**[Create Data Development Node]** > **[Virtual Node]** の順に選択します。



3. ノードタイプを **[Virtual Node]** に設定し、ノード名を入力します。ターゲットフォルダを選択し、**[Submit]** をクリックします。



4. ノードコードの編集**:** 仮想ノードでは、コードを編集する必要はありません。

5. ノードスケジューリングの設定

   ノードタスク編集エリアの右側にある **[Schedule]** をクリックし、**[node scheduling configuration]** ページへ移動します。 詳細は、「スケジューリングの設定」をご参照ください。

6. ノードを送信します。

   設定が完了した後、ページの左上隅にある **[Save]** をクリックするか、 または、**[Ctrl]** + **[S]** を押してノードを環境開発へ送信 (およびロックを解除) します。

7. ノードタスクを発行します。

   操作の詳細については、「リリース管理」をご参照ください。

8. 本番環境でテストします。

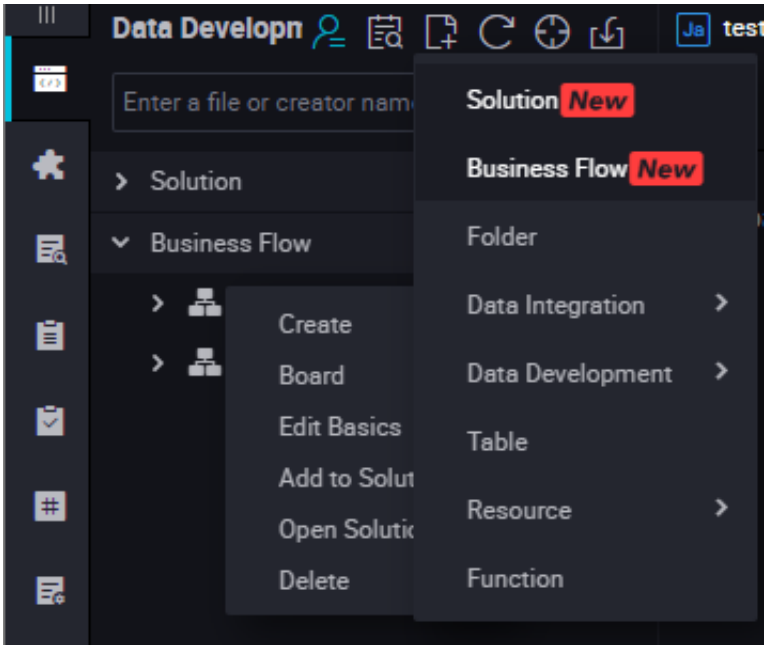
   操作の詳細については、「*#unique_46*」をご参照ください。

1.8.2 SQL コンポーネントノード

手順

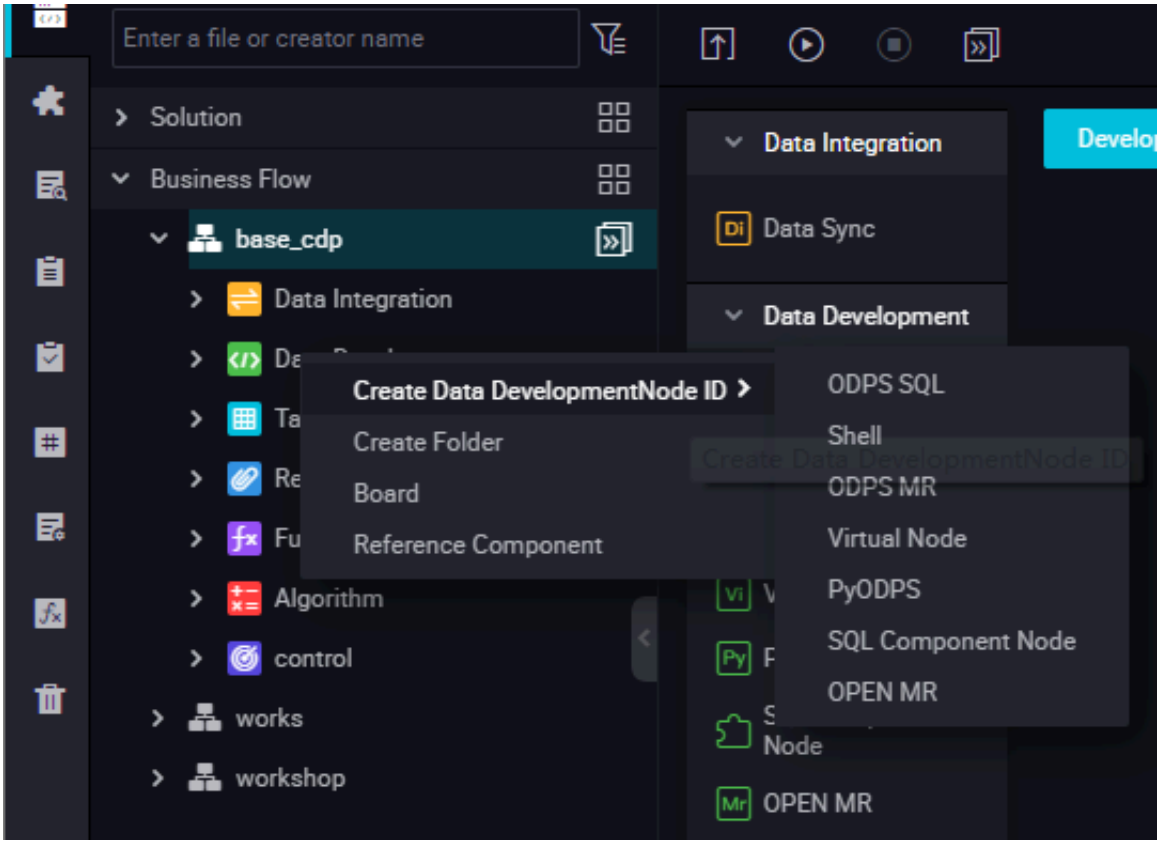
1. ビジネスフローを作成します。

   左側のナビゲーションウィンドウで **[Manual Business Flow]** 、**[Create Business Flow]** の順に選択します。



2. **SQL** コンポーネントノードを作成します。

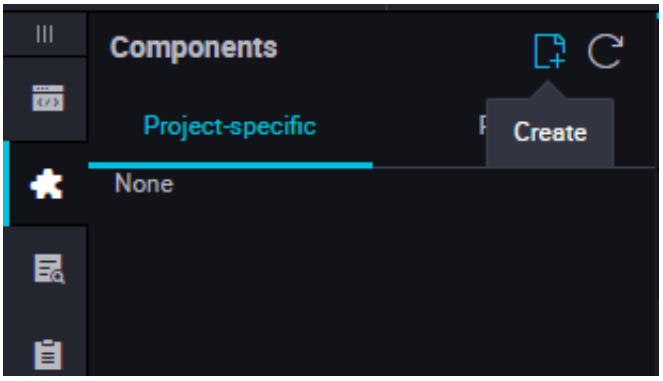
   [Data Development] を右クリックし、**[Create Data Development Node]** > **[SQL Component Node]** の順に選択します。



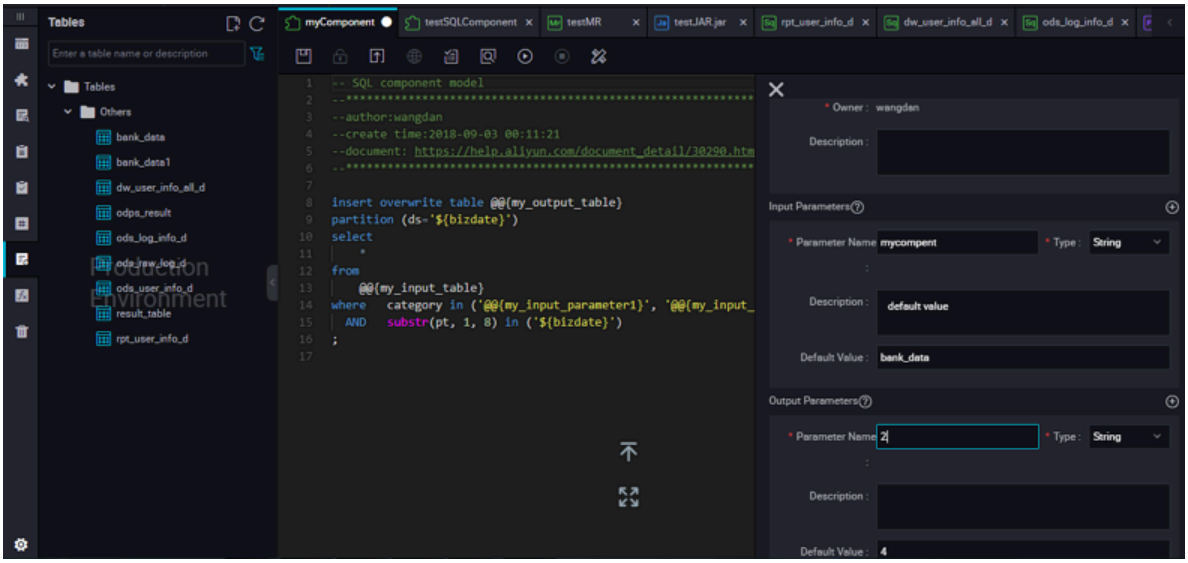
3. 開発効率を向上させるため、データタスクの開発者はプロジェクトメンバーとテナントメンバーによって提供されたコンポーネントを使い、データ処理ノードを作成します。

   - ローカルプロジェクトのメンバーが作成したコンポーネントは、**[Project Components]** にあります。
   - テナントメンバーが作成したコンポーネントは、**[Public Components]** にあります。

   ノードを作成する際、ノードタイプを 「**SQL component node**」 に設定し、ノード名を指定します。



   選択したコンポーネントのパラメーターを指定します。



   パラメーター名を入力し、パラメータータイプを「**Table**」または「**String**」に設定します。

   3 つの **"get_top_n"** パラメーターを順番に指定します。

   テーブルタイプ **(test_project.test_table)** のパラメーターに対して次の入力テーブルを指定します。

4. ノードスケジュールの設定

   ノードタスク編集エリアの右側で **[Schedule Configuration]** をクリックし、**[node scheduling configuration]** ページへ移動します。 詳細は、「スケジューリング設定」をご参照ください。

5. ノードを送信します。

   設定が完了した後、ページの左上隅にある **[Save]** をクリックするか、または **[Ctrl]** + **[S]** を押してノードを開発環境へ送信 (およびロックを解除) します。

6. ノードタスクを発行します。

   操作の詳細については、「 リリース管理」をご参照ください。

7. 本番環境でテストします。

   操作の詳細については、「*#unique_46*」をご参照ください。

SQL コンポーネントノードのバージョンのアップグレード

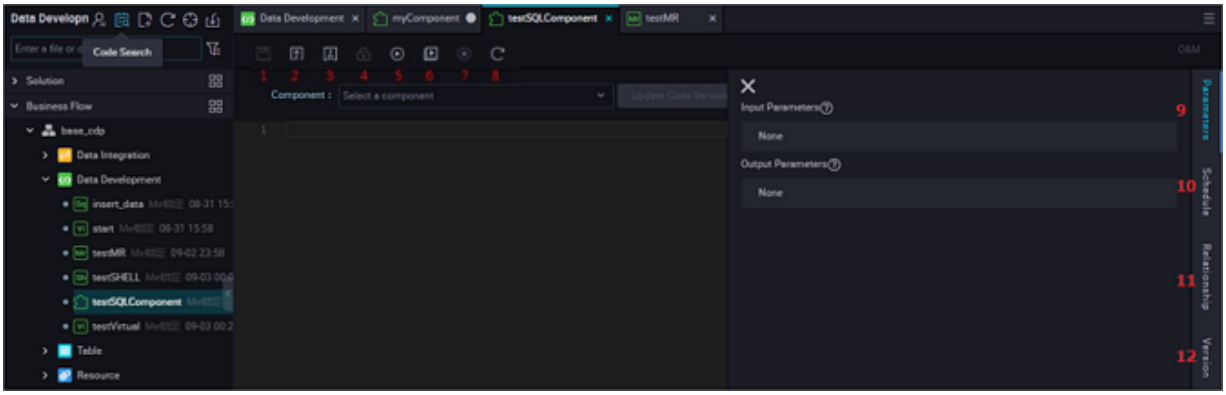
コンポーネントの開発者が新しいバージョンをリリースすると、コンポーネントユーザーはインスタンスで使用する既存のコンポーネントを最新版のコンポーネントにアップグレードするかどうかを選択できます。

コンポーネントのバージョンメカニズムにより、開発者は継続的にコンポーネントをアップグレードできます。また、コンポーネントユーザーは、コンポーネントのアップグレードによって改善されるプロセスの実行効率と最適化されたビジネス効果を継続的に享受できます。

たとえば、ユーザー **A** は、ユーザー **C** が開発した **v1.0** コンポーネントを使用しており、コンポーネントオーナー **C** がコンポーネントを **V2.0** にアップグレードしたとします。 アップグレード後もユーザー **A** は、 **v1.0** コンポーネントを使用し続けることができますが、アップグレードのリマインダーを受け取ります。 ユーザー **A** は旧コードと新コードを比較し、旧バージョンよりも新しいバージョンのビジネス効果の方が高いと感じた場合、最新版へアップグレードするかどうかを決定できます。

コンポーネントテンプレートに基づいて開発された **SQL** コンポーネントノードをアップグレードするには、**[Upgrade]** を選択して、**SQL** コンポーネントノードのパラメーター設定が新しいバージョンでも有効かどうかを確認するだけで済みます。新しいバージョンのコンポーネントの手順に沿って調整を行い、通常の **SQL** コンポーネントノードと同じように送信しリリースします。

インターフェイスの機能



インターフェイスの機能は次のとおりです。

No.	機能	説明
1	**Save**	クリックして、現在のコンポーネントの設定を保存します。
2	**Submit**	クリックして、現在のコンポーネントを開発環境へ送信します。
3	**Submit and Unlock**	クリックして、現在のノードを送信し、コードを編集できるようにロックを解除します。
4	**Steallock Edit**	現在のコンポーネントのオーナーではない場合、クリックし、ノードを **steallock** 編集します。
5	**Run**	クリックし、開発環境内でローカルでコンポーネントを実行します。
6	**Advanced Run (with Parameters)**	クリックして、コードに対して設定されたパラメーターを使って現在のノードのコードを実行します。 注： **Advanced Run** は、**Shell** ノードでは利用できません。
7	**Stop Run**	クリックし、実行中のコンポーネントを停止します。
8	**Re-load**	クリックして、インターフェイスを更新し、最後に保存した状態に戻します。 保存されていないコンテンツは失われます。 注： 設定センターでキャッシュを有効にしている場合、インターフェイスが更新された後、コードはキャッシュされますが保存はされません。 この場合、必要なバージョンを選択します。

No.	機能	説明
9	**Parameter Settings**	クリックし、コンポーネント情報、入力パラメーター設定、出力パラメーター設定を表示します。
10	**Attributes**	クリックして、ノードのオーナー、説明、パラメーター、リソースグループを設定します。
11	**Kinship**	**SQL** コンポーネントノード間の親子関係マップや、各 **SQL** コンポーネントノードの内部親子関係マップを表示します。
12	**Version**	現在のコンポーネントの送信および発行履歴を表示します。

1.8.3 ODPS MR ノード

MaxCompute では、**MapReduce** プログラミング **API** をサポートします。 **MapReduce** により提供される **Java API** を使って、**MaxCompute** でデータを処理するための **MapReduce** プログラムを作成することができます。 **ODPS MR** ノードを作成し、タスクスケジューリングで使用します。

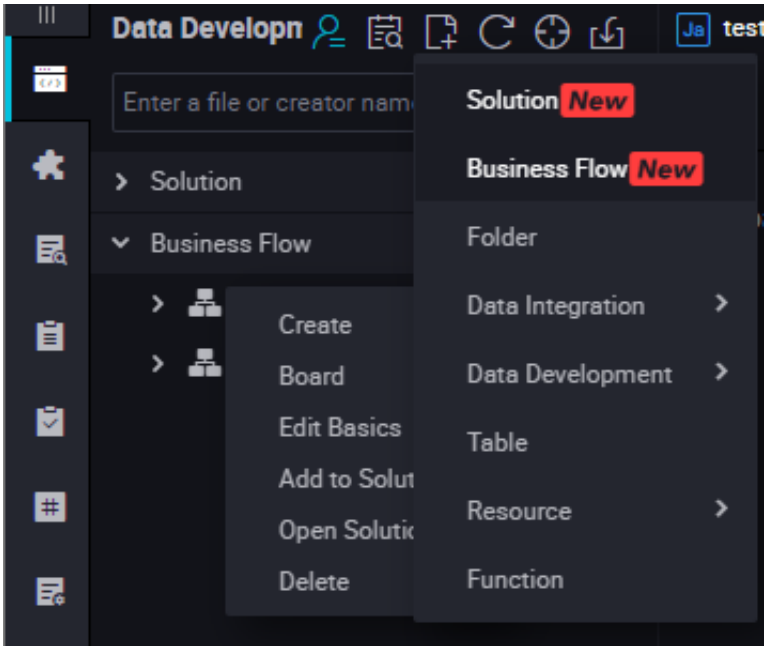
ODPS MR の編集方法や使用方法は、**MaxCompute** ドキュメントの「*WordCount* の例」に記載されている例をご参照ください。

ODPS MR ノードを使用するためには、使用するリソースを最初にアップロードしてリリースします。そして **ODPS MR** ノードを作成します。

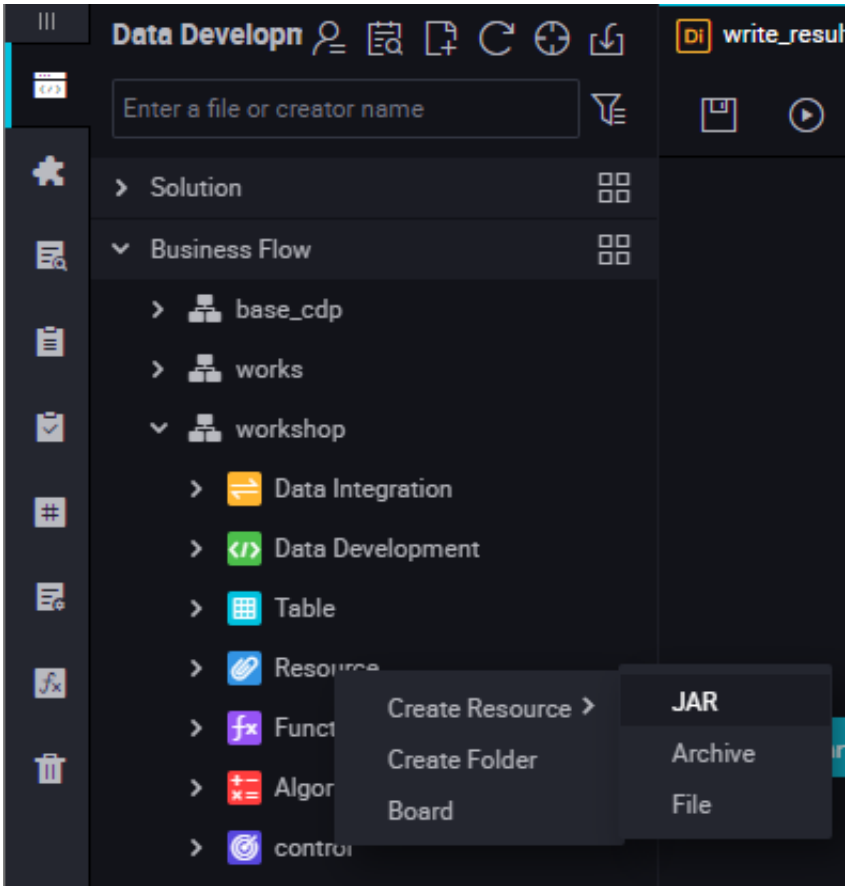
リソースインスタンスの作成

1. ビジネスフローを作成します。

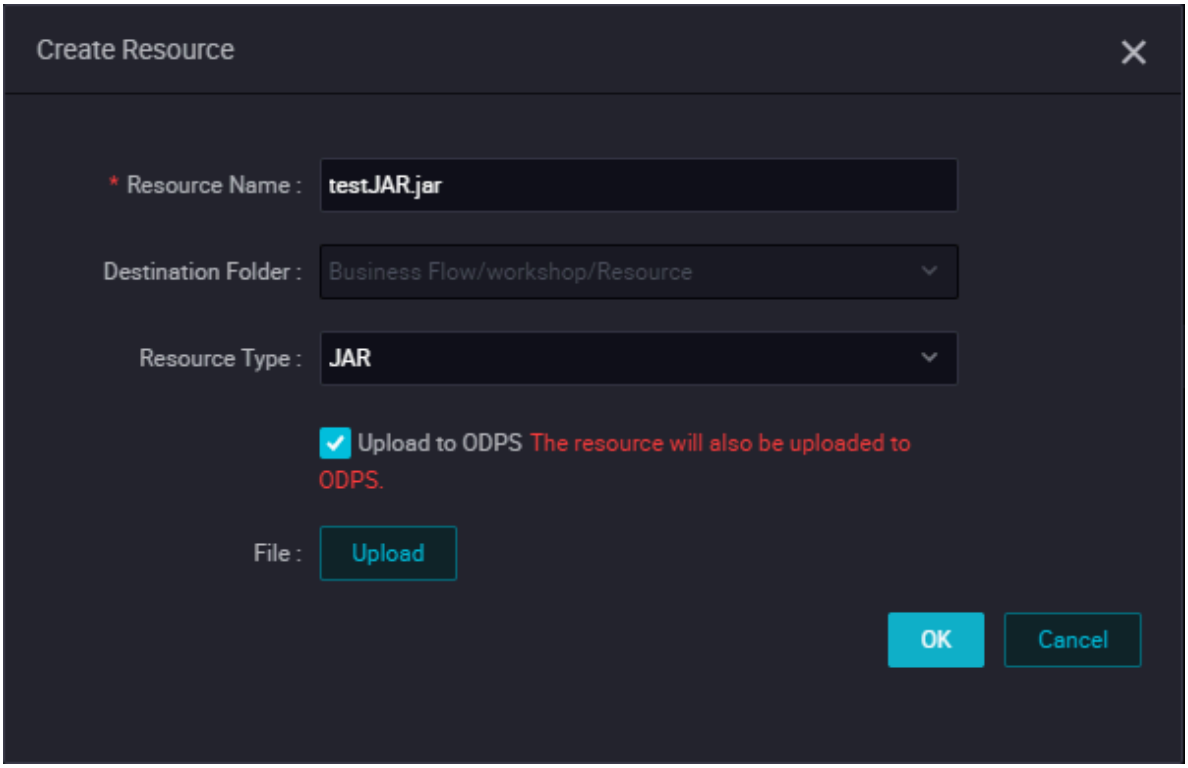
   左側のナビゲーションウィンドウで **[Manual Business Flow]** 、**[Create Business Flow]** の順に選択します。



2. **[Resource]** を右クリックし、**[Create Resource]** > **[jar]** の順に選択します。



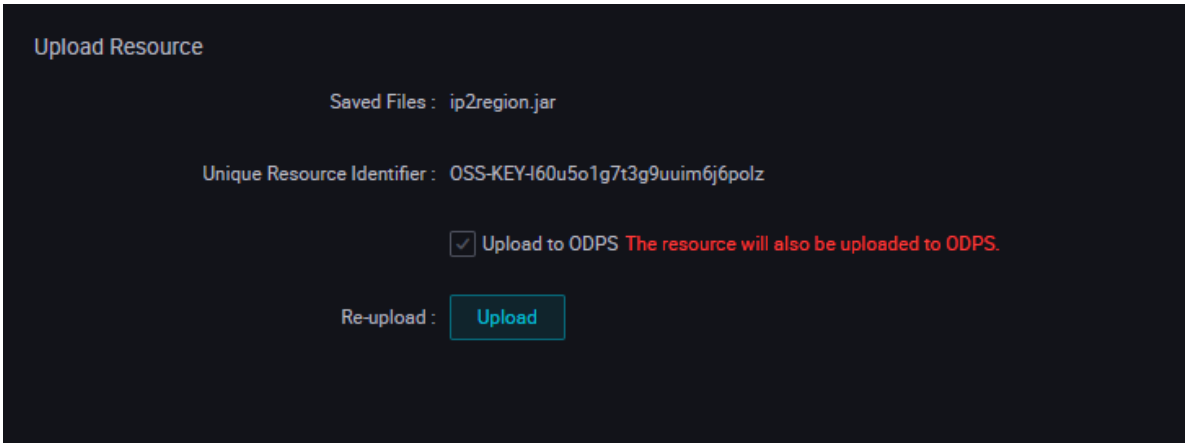
3. 命名規則に従い、**[Create Resource]** にリソース名を入力し、リソースタイプを「**jar**」に設定します。アップロードするローカル **jar** パッケージを選択します。



注：

- この **jar** パッケージが **ODPS** クライアントへアップロードされた後、**[Uploaded as the ODPS resource]** の選択を解除する必要があります。 このアップロードでは、リソースも **ODPS** へアップロードされます。 そうでない場合、アップロードプロセス中にエラーが報告されます。
- リソース名は、アップロードされるファイル名と同じにする必要はありません。
- リソース名の命名規則**:** 1 文字以上 **128** 文字以内の文字列で、英数字、アンダースコア、ドットを使用できます。 名前は大文字と小文字が区別されません。 リソースが **jar** リソースの場合、拡張子は **.jar** です。 リソースが **Python** リソースの場合、 拡張子は **.py** です。

4. **[Submit]** をクリックして、リソースを開発スケジューリングサーバーへ送信します。



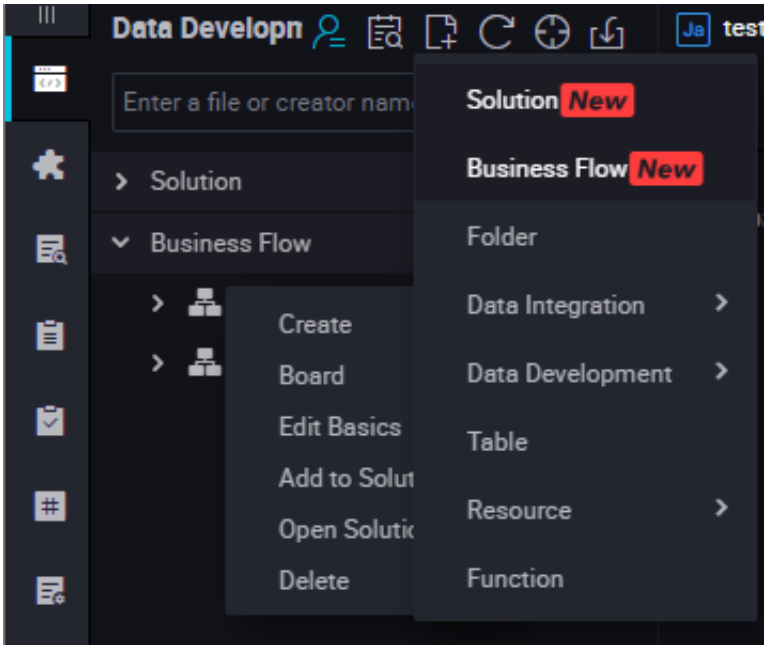
5. ノードタスクを発行します。

   操作の詳細については、「リリース管理」をご参照ください。

ODPS MR ノードの作成

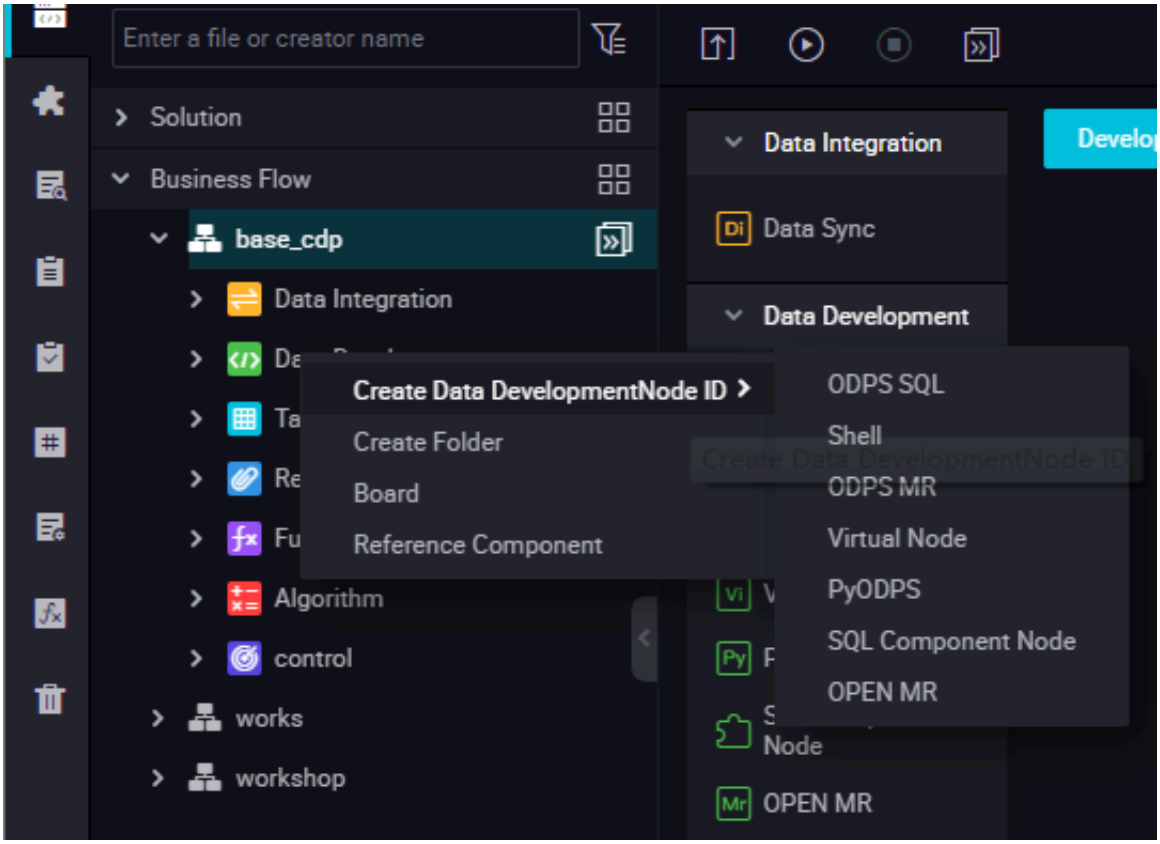
1. ビジネスフローを作成します。

   左側のナビゲーションウィンドウで **[Manual Business Flow]** 、**[Create Business Flow]** の順に選択します。

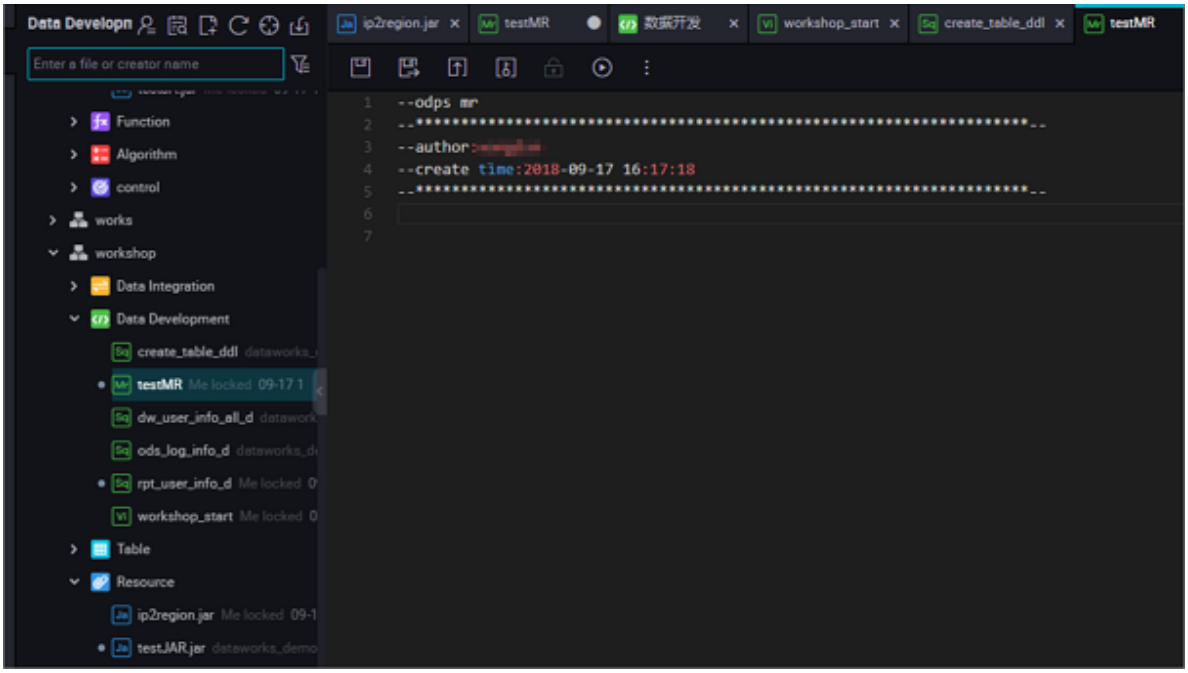


2. **ODPS MR** ノードを作成します。

   [Data Development] を右クリックし、**[Create Data Development Node]** > **[ODPS MR]** の順に選択します。



3. ノードコードを編集します。新しい **ODPS MR** ノードをダブルクリックし、次のインターフェイスに移動します。



ノードコードの編集例

```
jar -resources base_test.jar -classpath ./base_test.jar com.taobao.
edp.odps.brandnormalize.Word.NormalizeWordAll
```

コードの説明は次のとおりです。

- `-resources base_test.jar`: 参照先の **jar** リソースのファイル名を示します。
- `-classpath`: **jar** パッケージのパスです。参照リソースを右クリックし、このパスを取得します。

  > 注：
  > 新しい **ODPS MR** ノードをダブルクリックして、**ODPS MR** ノードインターフェイスに移動し、**jar** リソースを入力します。

- `com.taobao.edp.odps.brandnormalize.Word.NormalizeWordAll`: 実行時に呼び出される **jar** パッケージのメインクラスを示します。 **jar** パッケージのメインクラス名と同じにする必要があります。

1 つの **MR** から複数の **jar** リソースを呼び出す場合、クラスパスは`-classpath ./xxxx1.jar,./xxxx2.jar` のとおりに記述する必要があります。**2** つのパスはコンマで区切ります。

4. ノードスケジューリングの設定

   ノードタスク編集エリアの右側で **[Schedule]** をクリックし、**[node scheduling configuration]** ページへ移動します。 詳細は、「スケジューリング設定」をご参照ください。

5. ノードを送信します。

   設定が完了した後、ページの左上隅にある **[Save]** をクリックするか、または **[Ctr]l + [S]** を押して ノードを開発環境へ送信 (およびロックを解除) します。

6. ノードタスクを発行します。

   操作の詳細については、「 リリース管理」をご参照ください。

7. 本番環境でテストします。

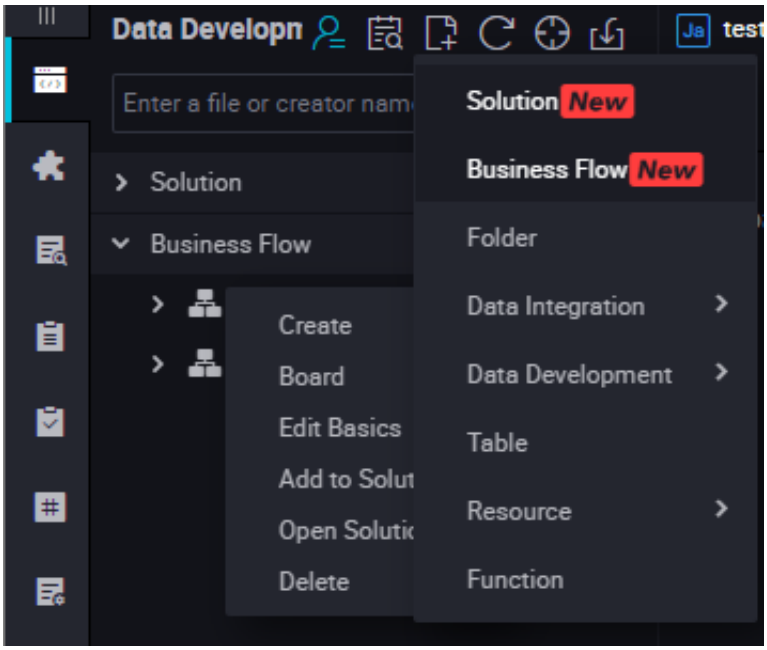
   操作の詳細については、「*#unique_46*」をご参照ください。

1.8.4 マニュアルデータ統合ノード

現在、データ統合 **(data Integration)** タスクで使用できるデータソースは、**MySQL**、**DRDS**、**SQL Server**、 **PostgreSQL**、 **Oracle**、 **MongoDB**、 **DB2**、**Table Store**、 **OTSStream**、**OSS**、 **FTP**、 **Hbase**、 **LogHub**、 **HDFS**、および **Stream** です。 サポートされているデータソースについての詳細は、「*#unique_17*」をご参照ください。

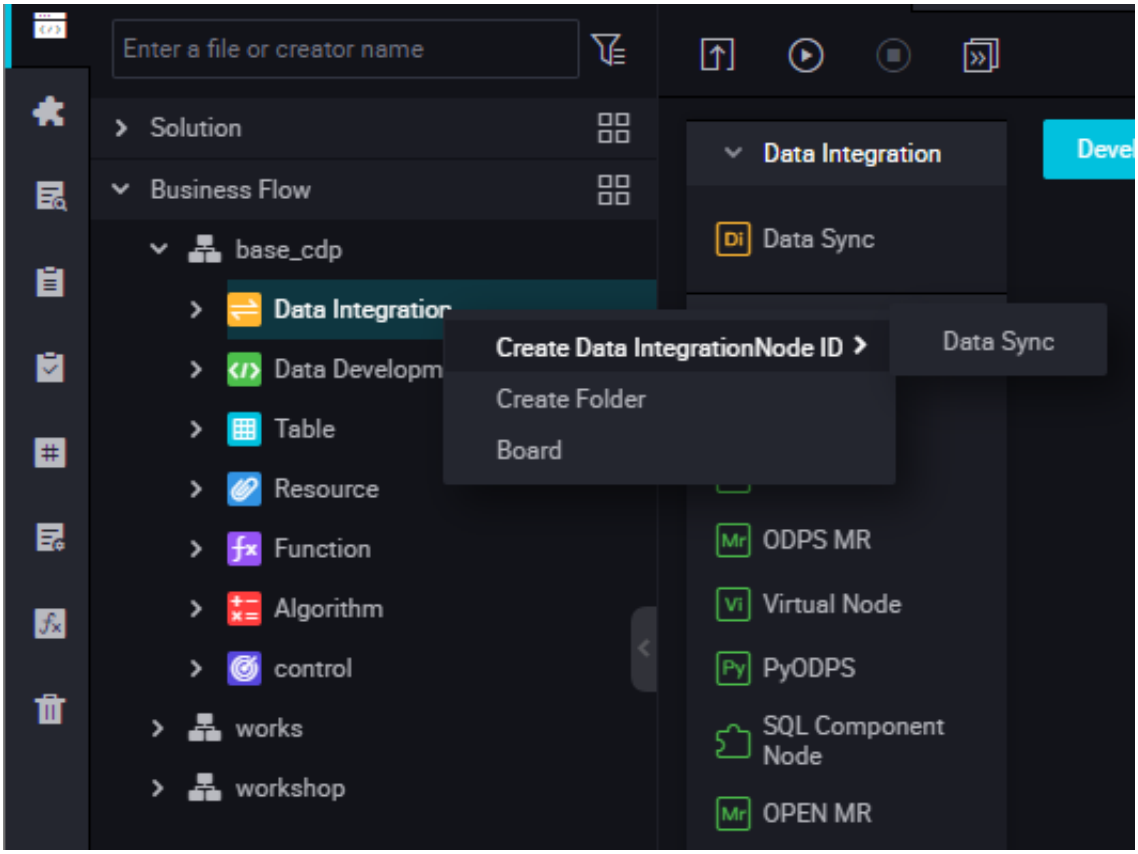
1. ビジネスフローを作成します。

   左側のナビゲーションウィンドウで、**[Manual Business Flow]**、**[Create Business Flow]** の順に選択します。



2. データ統合ノードを作成します。

   [Data Integration] を右クリックし、**[Create Data Data Integration Node]** > **[Data Integration]** の順に選択します。



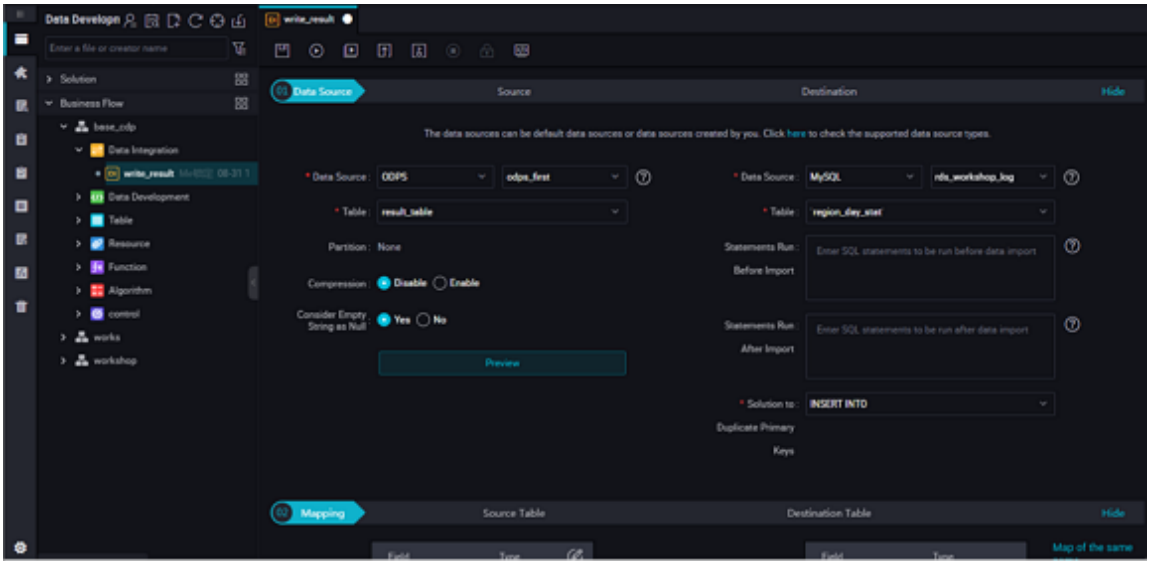
3. 統合タスクを設定します。

   ソーステーブル名とターゲットテーブル名を入力すると、タスクの簡易設定が完了します。

   テーブル名を入力すると、そのテーブル名と一致するオブジェクトのリストが自動的に表示されます (現在、完全にテーブル名が一致するオブジェクトのみが表示されるため、 正しく完全なテーブル名を入力する必要があります)。 現在の統合センターではサポートされていないオブジェクトは **[Not supported]** と表示されます。 オブジェクトにマウスを移動します。 データベース、**IP** アドレス、テーブルのオーナーといったオブジェクトの詳細情報が自動的に表示されます。 適切なテーブルオブジェクトを選択するのに役立ちます。 オブジェクトを選択

したら、そのオブジェクトをクリックします。 列の情報が自動的に入力されます。 移動、削除、追加といった列の編集を行うことができます。

a. 統合テーブルを設定します。



b. データソースを編集します。

通常、必要がない限り、ソーステーブルのコンテンツを編集する必要はありません。

- 新しい列を挿入するには、列の右側にある **[Insert]** をクリックします。
- 列を削除するには、、列の右側にある **[Delete]** をクリックします。

c. データの統合先の情報を編集します。

通常、必要がない限り統合先のテーブルのフィールド情報を編集する必要はありません。

> 注：
> 統合先が **ODPS** テーブルの場合、列を削除することはできません。 統合タスクの設定では、ソーステーブルのフィールド設定は、フィールド名ごとではなくページごとに、統合先のテーブル設定と **1** 対 **1** で合致しています。

d. 増分統合とフル統合

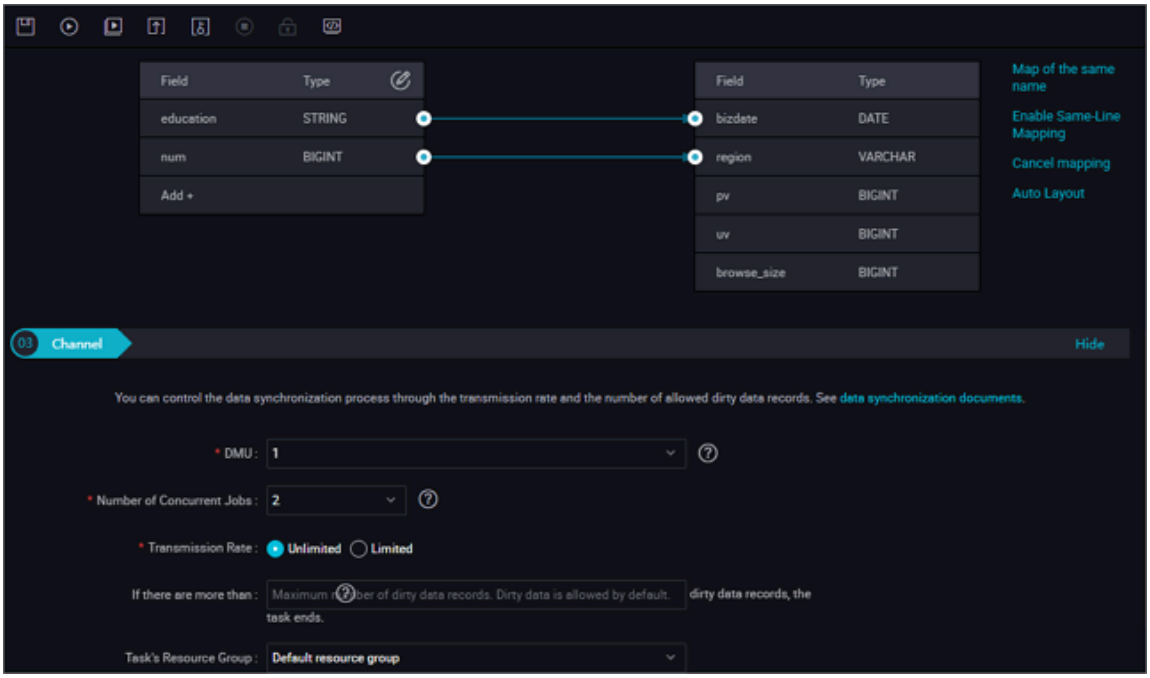
- 増分統合のシャード形式**: ds=${bizdate}**
- フル統合のシャード形式**: ds=***

> 注：
> 複数のシャードを同期する必要がある場合、統合センターは簡易正規表現をサポートします。

- たとえば、複数のシャードを同期する必要があるが、正規表現の記述が難しい場合は、メソッド`ds=20180312 | ds=20180313 | ds=20180314;`を使用します。
- シャードを同じ範囲内で同期する必要がある場合、統合センターは、`/*query*/ds>=20180313 and ds<20180315;`といった拡張構文をサポートします。このメソッドを使用する場合、**/query/** を追加します。
- 変数 **bizdate** は、パラメーター`-p"-Dbizdate=$bizdate -Denv_path=$env_path -Dhour=$hour"`で定義する必要があります。 たとえば、`pt=${selfVar}`といった変数をカスタマイズする場合、パラメーターの変数を`-p"-Dbizdate=$bizdate -Denv_path=$env_path -Dhour=$hour -DselfVar=xxxx"`のように定義します。

e. フィールドのマッピング

フィールドは、フィールド名やフィールドタイプではなく、ソーステーブルと統合先のテーブルのフィールド位置に基づいてマッピングされます。



注：

ソーステーブルが **ODPS** テーブルの場合、データの統合中にフィールドを追加することはできません。 ソーステーブルが **ODPS** テーブル以外の場合、データの統合中にフィールドを追加することができます。

f. トンネル制御

トンネル制御機能は、統合タスクを選択する際の速度やエラー率を制御するために使用します。

- **DMU:** データ統合中に消費されるリソース **(CPU**、メモリ、ネットワーク帯域幅など**)** を測定するデータ移行単位
- **Concurrent job count:** データ統合タスクでデータ記憶媒体へデータの書き込み、データ記憶媒体からのデータの読み込みを同時に行うために使用されるスレッドの最大数
- **integration speed:** 統合タスクの最高速度
- **Maximum error count:** ダーティデータ量を制御するために使用します。ソース元テーブルのフィールドタイプが統合先のフィールドタイプと一致しない場合、同期データ量に基づいてユーザーが設定します。 許容されるダーティデータカウントの最大数を指定します。 「**0**」に設定すると、ダーティデータは一切許容されません。指定していない場合、ダーティデータは許容されます。
- **Task resource group:** 現在の統合ノードがあるリソースグループを選択するには、データ統合ページでリソースグループを追加または編集します。

4. ノードスケジューリングの設定

ノードタスク編集エリアの右側にある **[Schedule]** をクリックし、**[node scheduling configuration]** ページへ移動します。 詳細は、「スケジューリングの設定 」をご参照ください。

5. ノードタスクを送信します。

設定が完了した後、ページの左上隅にある **[Save]** をクリックするか、または **[Ctrl]** + **[S]** を押してノードを開発環境へ送信 **(**およびロックを解除**)** します。

6. ノードタスクを発行します。

操作の詳細については、「リリース管理」をご参照ください。

7. 本番環境でテストします。

操作の詳細については、「*#unique_20*」をご参照ください。

1.8.5 PyODPS ノード

DataWorks では、**PyODPS** タスクも実行することができ、**MaxCompute** の **Python SDK** と統合されます。 **Python** コードを直接編集して、**DataWorks** の **PyODPS** ノードで **MaxCompute** を操作することができます。

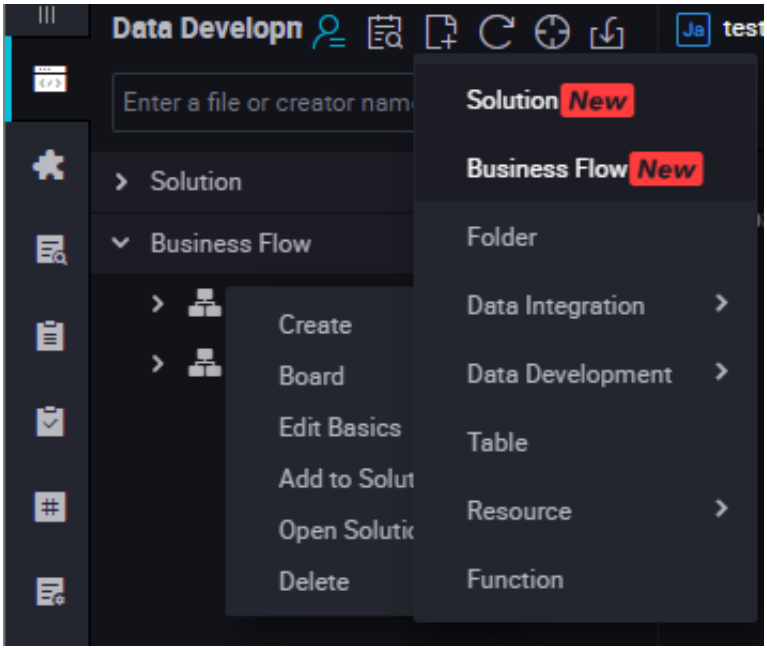
PyODPS ノードの作成

MaxCompute には、**MaxCompute** の操作に使用できる *Python SDK* が実装されています。

PyODPS ノードを作成するには、次の手順に従います。

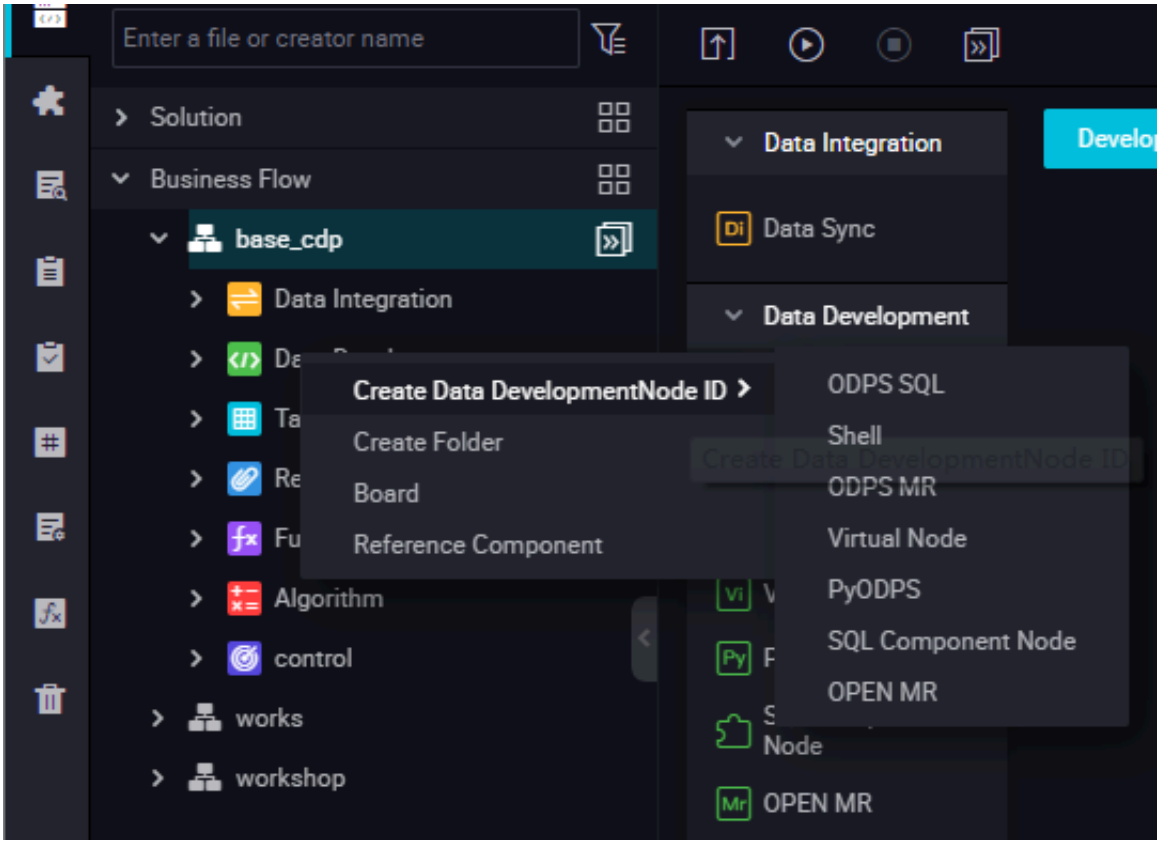
1. ビジネスフローを作成します。

   左側のナビゲーションウィンドウで**[Manual Business Flow]** 、**[Create Business Flow]** の順にを選択します。



2. **PyODPS** ノードを作成します。

   [Data Development] を右クリックし、**[Create Data Development Node]** > **[PyODPS]** の順に選択します。



3. **PyODPS** ノードを編集します。

   a. **ODPS** ポータル

   DataWorks では、**PyODPS** ノードには、**ODPS** エントリであるグローバル変数 **"odps"** または **"o"** が含まれています。手動で **ODPS** エントリを定義する必要はありません。

   ```
   print(odps.exist_table('PyODPS_iris'))
   ```

   b. **SQL** 文を実行します。

   PyODPS では、**ODPS SQL** クエリをサポートしており、実行結果を読み取ることができます。 **execute_sql** または **run_sql** メソッドのリターン値は実行中のインスタンスです。

   注：
   ODPS コンソールで実行できるコマンドのすべてが、**ODPS** で受け入れられる **SQL** 文ということではありません。 **DDL** 文や **DML** 文以外を呼び出すには、別のメソッドを使用する必要があります。 たとえば、**GRANT** 文や **REVOKE** 文を呼び出すに

は、**"run_security_query"** メソッドを使用します。**PAI** コマンドを呼び出すには、**"run_xflow"** または **"execute_xflow"** メソッドを使用します。

```
o.execute_sql('select * from dual') # Run the SQL statements in
synchronous mode. Blocking continues until execution of the SQL
statement is completed.
instance = o.runsql('select * from dual') # Run the SQL
statements in asynchronous mode.
print(instance.getlogview_address()) # Obtain the logview address
.
instance.waitforsuccess() # Blocking continues until execution of
 the SQL statement is completed.
```

c. ランタイムパラメーターを設定します。

ランタイムパラメーターの設定が必要になる場合があります。パラメータータイプ **dict** を使って、ヒントパラメーターを設定します。

```
o.execute_sql('select * from PyODPS_iris', hints={'odps.sql.mapper
.split.size': 16})
```

sql.setings をグローバル設定に追加すると、関連するランタイムパラメーターが、各 **running.python** に追加されます。

```
from odps import options
options.sql.settings = {'odps.sql.mapper.split.size': 16}
o.execute_sql('select * from PyODPS_iris') # "hints" is added
based on the global configuration.
```

d. SQL 文の実行結果を読み取ります。

SQL 文を実行するインスタンスでは、**open_reader** 操作が直接実行されます。あるケースでは、構造化されたデータが **SQL** 文の実行結果として返されます。

```
with odps.execute_sql('select * from dual').open_reader() as
reader:
for record in reader: # Process each record.
```

別のケースでは、**SQL** 文で **desc** が実行される場合があります。この場合、オリジナルの **SQL** 文の実行結果は、**reader.raw** 属性を使って取得されます。

```
with odps.execute_sql('desc dual').open_reader() as reader:
print(reader.raw)
```

注：

ユーザー定義のスケジューリングパラメーターは、データ開発に使用されます。**PyODPS** ノードがページ上で直接トリガーされる場合、時間を明確に指定する必要があります。**PyODPS** ノードの時間は、**SQL** ノードの時間のように直接置換することができません。

4. ノードスケジューリングの設定

   ノードタスク編集エリアの右側で **[Schedule]** をクリックし、**[node scheduling configuration]** ページへ移動します。 詳細は、「スケジューリングの設定」をご参照ください。

5. ノードを送信します。

   設定が完了した後、ページの左上隅の **[Save]** をクリックするか、または **[Ctr]l + [S]** を押してノードを開発環境へ送信 (およびロックを解除) します。

6. ノードタスクを発行します。

   操作の詳細については、「リリース管理」をご参照ください。

7. 本番環境でテストします。

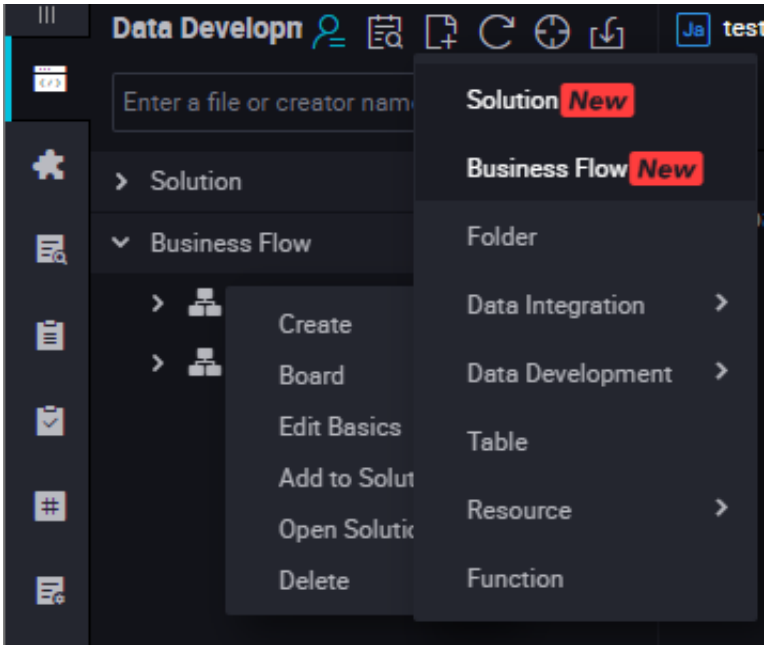
   操作に関する詳細は、「*#unique_46*」をご参照ください。

1.8.6 ODPS SQL ノード

ODPS SQL の構文は **SQL** 構文と似ています。**ODPS SQL** 構文は、データ量は膨大 **(TB** 級) だが、リアルタイム要件が高くない分散シナリオに適用されます。 スループット重視の **OLAP** アプリケーションです。 ジョブの準備から送信までのプロセスが完了するのに長時間がかかるため、**ODPS SQL** は数万単位でのトランザクションを処理する必要のあるビジネスに対して推奨されます。

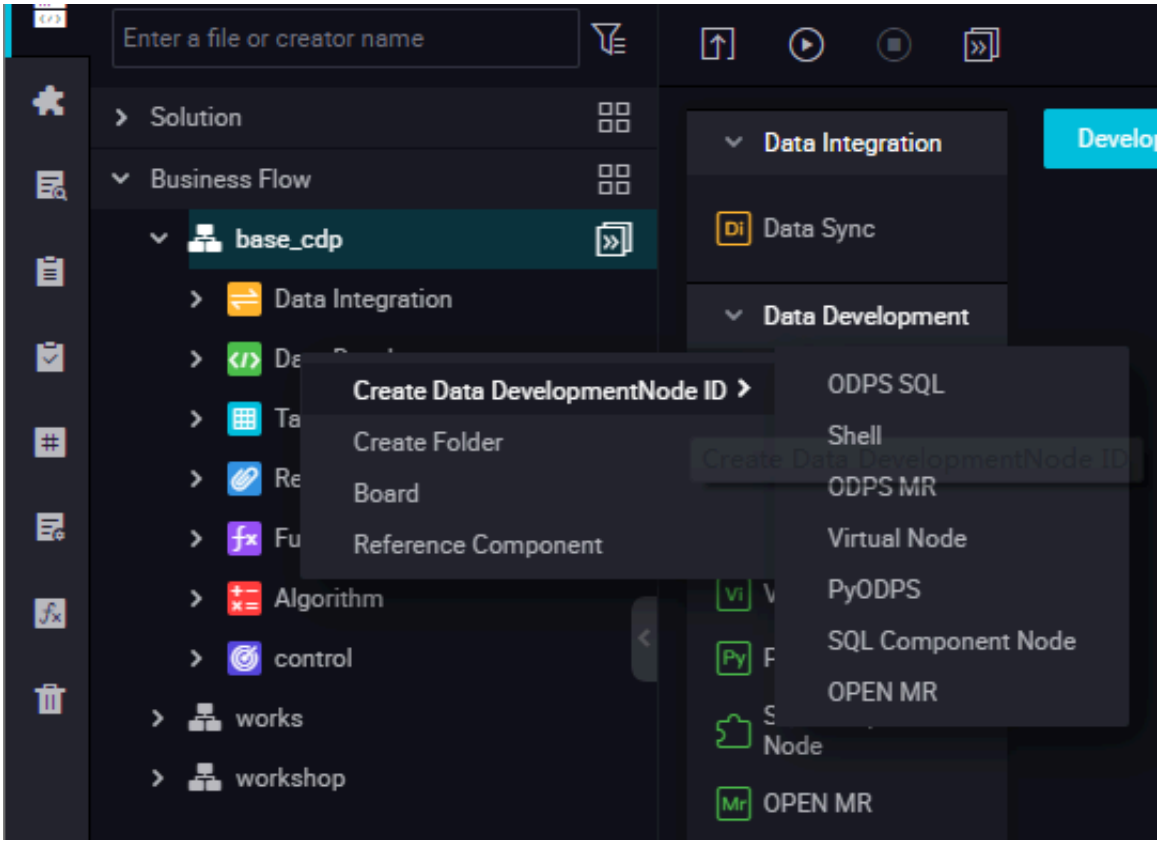
1. ビジネスフローを作成します。

   左側のナビゲーションウィンドウで **[Manual Business Flow]**、**[Create Business Flow]** の順に選択します。



2. **ODPS SQL** ノードを作成します。

   [Data Development] を右クリックし、**[Create Data Development Node]** > **[ODPS SQL]** の順に選択します。



3. ノードコードを編集します。

   SQL 構文に関する詳細については、「*MaxCompute SQL* 文」をご参照ください。

4. ノードスケジューリングの設定

   ノードタスク編集領域の右側にある **[Schedule]** をクリックし、**[node scheduling configuration]** ページへ移動します。 詳細は、「スケジューリングの設定」 をご参照ください。

5. ノードを送信します。

   設定が完了した後、ページの左上隅にある **[Save]** をクリックするか、または **[Ctrl]** + **[S]** を押して開発環境へノードを送信 (ロックを解除) します。

6. ノードタスクを発行します。

   操作の詳細については、「リリース管理」をご参照ください。

7. 本番環境でテストします。

   操作の詳細については、「*#unique_46*」をご参照ください。

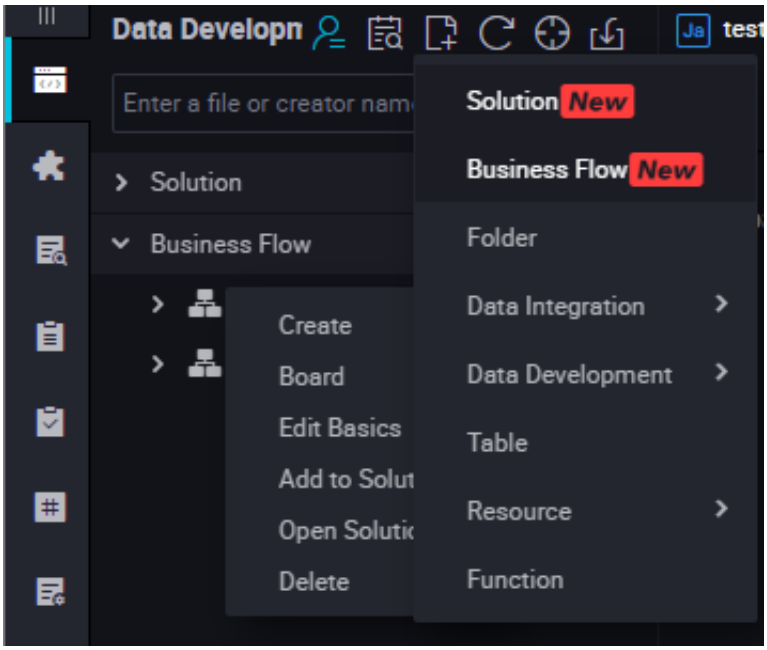
1.8.7 SHELL ノード

SHELL タスクは、標準の **SHELL** 構文をサポートしていますが、対話型構文はサポートしていません。 **SHELL** タスクは、デフォルトリソースグループで実行されます。 **IP** アドレスやドメイン名にアクセスする場合、**[Project Configuration]** を選択し、**IP** アドレスまたはドメイン名をホワイトリストへ追加します。

手順

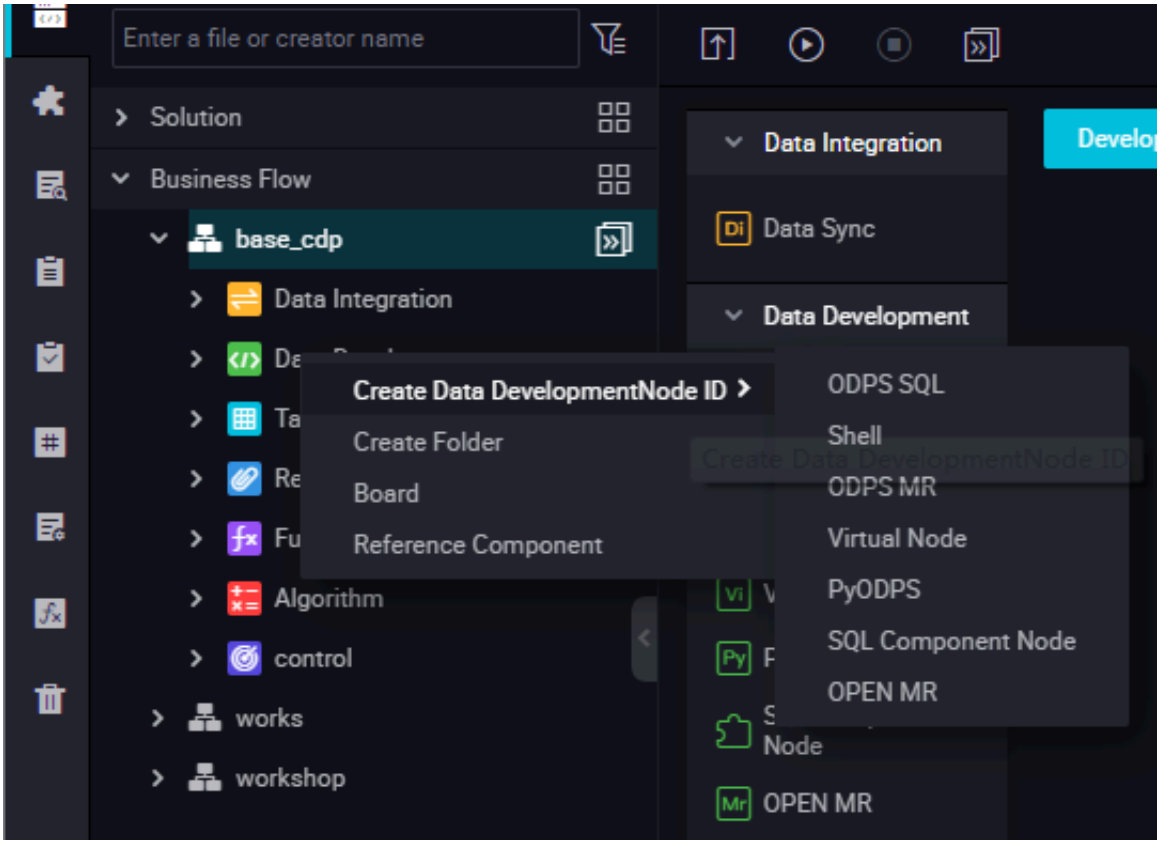
1. ビジネスフローを作成します。

   左側のナビゲーションウィンドウで **[Manual Business Flow]** 、**[Manual Business Flow]** の順に選択します。



2. **SHELL** ノードを作成します。

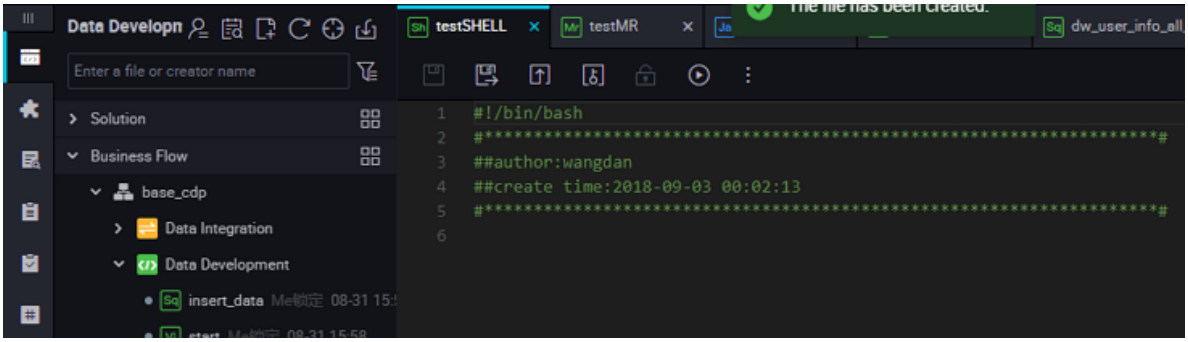
   [Data Development] を右クリックし、**[Create Data Development Node]** > **[SHELL]** の順に選択します。



3. ノードタイプを「**SHELL**」に設定し、ノード名を入力します。ターゲットフォルダを選択し、**[Submit]** をクリックします。

4. ノードコードを編集します。

   SHELL ノードコードの編集ページへ移動し、コードを編集します。

   SHELL 文でシステムスケジューリングパラメーターを呼び出す場合、次のとおり **SHELL** 文を記述します。

   ```
   echo "$1 $2 $3"
   ```

   > 注：
   > **Parameter 1 Parameter 2...** 複数のパラメーターはスペースで区切ります。 システムスケジューリングパラメーターの使用に関する詳細は、「パラメーターの設定」をご参照ください。

5. ノードスケジューリングの設定

   ノードタスク編集エリアの右側の **[Schedule]** をクリックし、**[node scheduling configuration]** ページへ移動します。 詳細は、「スケジューリングの設定」をご参照ください。

6. ノードを送信します。

   設定が完了した後、ページの左上隅にある **[Save]** をクリックするか、または **[Ctr]l** + **[S]** を押してノードを開発環境へ送信 (およびロックを解除) します。

7. ノードタスクをリリースします。

   操作の詳細については、「リリース管理」をご参照ください。

8. 本番環境でテストします。

   操作の詳細については、「*#unique_46*」をご参照ください。

使用例

SHELL を使ってデータベースへ接続します。

- **Alibaba Cloud** でデータベースを構築し、リージョンが中国 (上海) の場合、次のホワイトリスト内の **IP** アドレスを使用してデータベースを開き、接続します。

  10.152.69.0/24、10.153.136.0/24、10.143.32.0/24、120.27.160.26、10.46.67.156、120.27.160.81、10.46.64.81、121.43.110.160、10.117.39.238、121.43.112.137、10.117.28.203、118.178.84.74、10.27.63.41、118.178.56.228、10.27.63.60、118.178.59.233、10.27.63.38、118.178.142.154、10.27.63.15、100.64.0.0/8

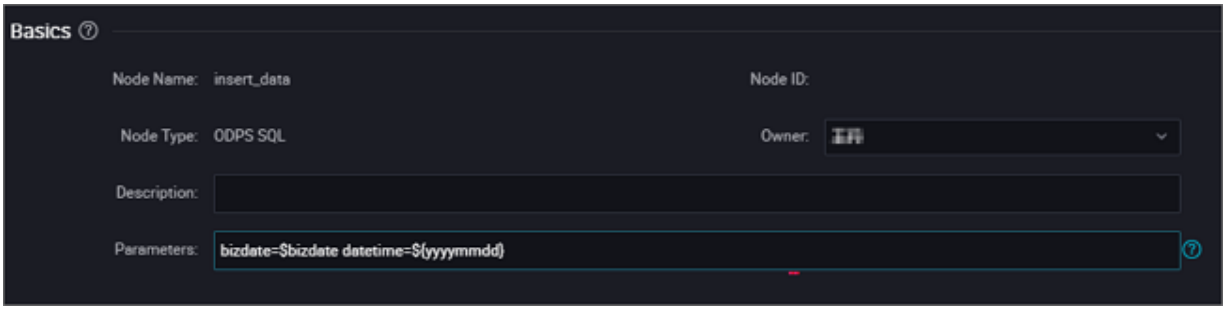
  注：
  Alibaba Cloud でデータベースを構築したが、リージョンが中国 (上海) ではない場合、インターネットを使用するか、またはデータベースと同じリージョンで **ECS** インスタンスをスケジューリングリソースとして購入し、カスタムリソースグループで **SHELL** タスクを実行することを推奨します。
- データベースがローカルで構築されている場合、インターネット接続を使い、上記のホワイトリスト内の **IP** アドレスでデータベースを開くことを推奨します。

  注：
  カスタムリソースグループを使って **SHELL** タスクを実行する場合、カスタムリソースグループ内のマシンの **IP** アドレスを上記ホワイトリストに追加する必要があります。

1.9 マニュアルタスクパラメーターの設定

1.9.1 基本属性

次の図は、基本属性の設定インターフェイスを示しています。



- **Node Name:** ワークフローノードを作成する際に入力するノード名です。 ノード名を編集するには、ディレクトリツリーでノードを右クリックし、ショートカットメニューから **[Rename]** を選択します。
- **Node ID:** タスクの送信時に生成される一意のノード **ID** で、編集はできません。
- **Node ID:** タスクの送信時に生成される一意のノード **ID** で、編集はできません。

- **Owner:** ノードのオーナーです。 デフォルトでは現在ログインしているユーザーが、新しく作成したノードのオーナーになります。 オーナーを変更するには、入力ボックスをクリックして、オーナーの名前を入力するか、または他のユーザーを選択します。

   注：
  他のユーザーを選択する場合、そのユーザーは現行プロジェクトのメンバーである必要があります。

- **Description:** 通常、ビジネスとノードの目的を説明するために使用します。
- **Parameter:** タスクスケジューリングの際、コードの変数に値を割り当てるために使用します。

  たとえば、変数 **pt=${datetime}** を使ってコード内で時間を指定する場合、ここで変数に値を割り当てます。 割り当てる値には、スケジューリング組み込み時間パラメーター **"datetime=$bizdate"** を使用できます。
- **Resource Group:** ノードを実行するためのリソースグループを指定します。

各種ノードタイプのパラメーター値の割り当て形式

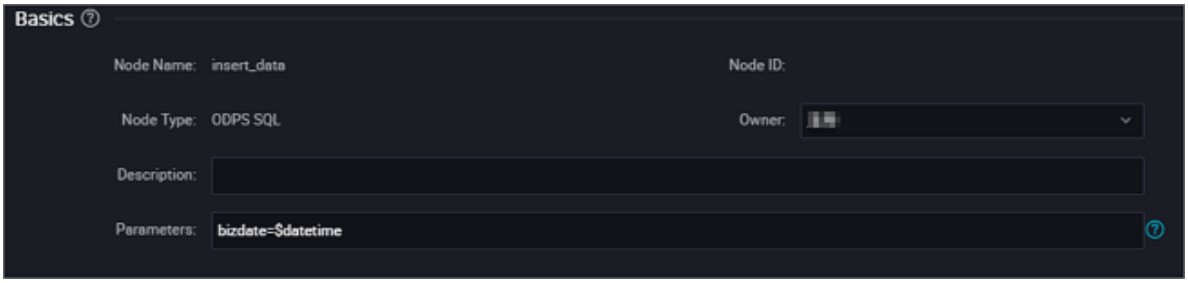
- **ODPS SQL、ODPSPL、ODPS MR、XLIB** タイプ: `Variable name 1=Parameter 1 Variable name 2=Parameter 2...` 複数のパラメーターはスペースで区切ります。
- **SHELL** タイプ: `Parameter 1 Parameter 2...` 複数のパラメーターはスペースで区切ります。

頻繁に使用される時間パラメーターのいくつかは、組み込みスケジューリングパラメーターとして提供されています。 パラメーターの詳細については、「パラメーターの設定」をご参照ください。

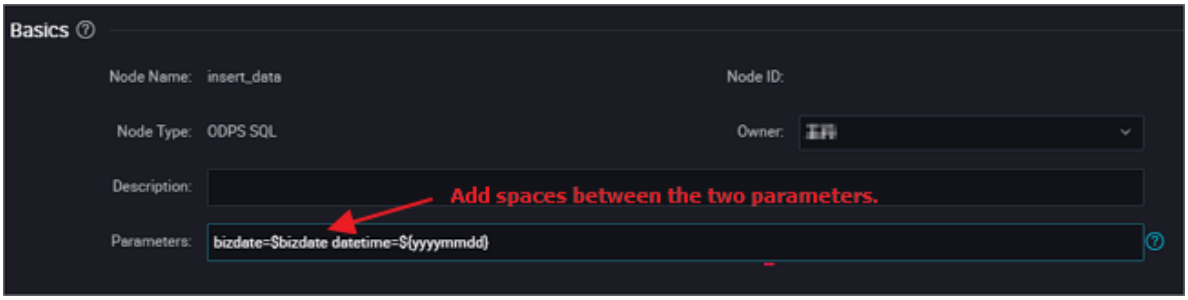
1.9.2 マニュアルノードパラメーターの設定

自動的にスケジュールされた時間にタスクが実行される際、動的に環境の変化に適用できるようにするために、**DataWorks** ではパラメーター設定機能を提供します。 パラメーターを設定する前に次の問題点について注意してください。

- パラメーター内の "=" の両端にはスペースを追加することができません。 正しい: **bizdate=$bizdate**



- 複数のパラメーター (存在する場合) は、スペースで区切ります。



システムパラメーター

DataWorks では、2 つのシステムパラメーターを提供し、次のとおり定義されます。

- **${bdp.system.cyctime}:** インスタンスのスケジュール実行時間を定義します。 デフォルト形式**: yyyymmddhh24miss**
- **${bdp.system.bizdate}:** インスタンスが計算される営業日を定義します。 デフォルトの営業日は、実行日の前日です。デフォルトの表示形式は、**yyyymmdd** です。

定義に応じて、ランタイムと営業日の計算式は次のとおりです。`Runtime = Business date - 1`

システムパラメーターを使用するためには、編集ボックスでシステムパラメーターを設定せずにコード内の **${bizdate}** を直接参照します。システムは自動的にコード内のシステムパラメーターの参照フィールドを置換します。

 注:

周期タスクのスケジューリング属性は、スケジュールランタイムを使って設定されます。 したがって、インスタンスのスケジュールランタイムをバックトラックしたり、インスタンスのシステムパラメーター値を取得したりできます。

例

ODPS_SQL タスクを毎日 **00:00** から **23:59** 間の毎時間実行するように設定します。 コードでシステムパラメーターを使用するには、次の文を実行します。

```
insert overwrite table tb1 partition(ds ='20180606') select
c1,c2,c3
from (
select * from tb2
where ds ='${bizdate}');
```

非 Shell ノードのスケジューリングパラメーターの設定

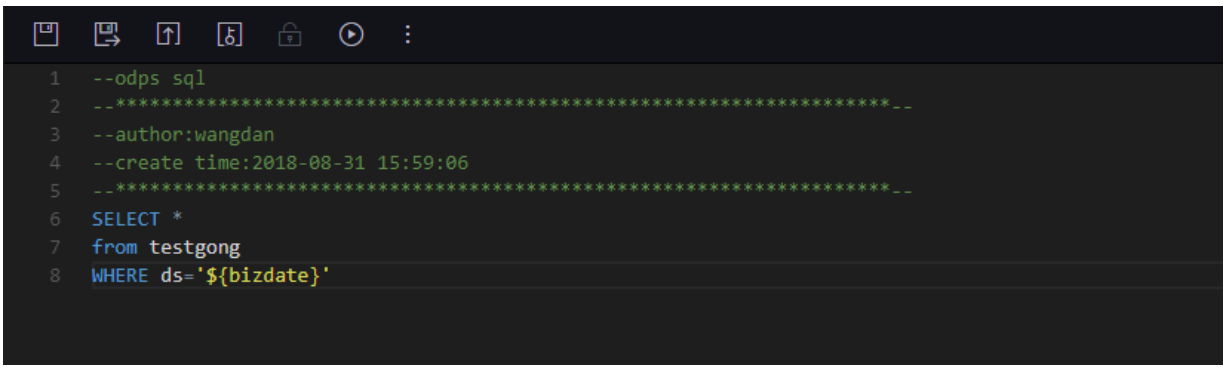
注：

SQL コードの変数名には、**a-z**、**A-Z**、数字、アンダースコアを使用できます。 変数名が **date** の場合、値 **$bizdate** は自動的にこの変数に割り当てられます。スケジューリングパラメーターの設定で値を割り当てる必要はありません。 もし他の値が割り当てられていても、デフォルトで値 **$bizdate** が自動的に割り当てられるため、コードではこの値は使用されません。

非 **Shell** ノードでは、**${variable name}** をコードに追加します (この関数が参照されていることを示します)。 特定の値を入力し、値をスケジューリングパラメーターへ割り当てます。

たとえば、**ODPS SQL** ノードに対しては、**${variable name}** をコードに追加します。ノードのパラメーター項目 "変数名 = ビルトインスケジューリングパラメーター" を設定します。

コード内で参照されているパラメーターでは、スケジューリング中の解析値を追加する必要があります。



Shell ノードのスケジューリングパラメーターの設定

Shell ノードのパラメーター設定手順は、非 **Shell** ノードのパラメーター設定手順と似ていますが、規則が異なります。 **Shell** ノードでは、変数名はカスタマイズできず、**$1,$2,$3...** という名前である必要があります。

たとえば、**Shell** ノードの場合、コード内の **Shell** 構文宣言は **$1**、スケジューリングのノードパラメーター設定は、**$xxx** (ビルトインスケジューリングパラメーター) です。 つまり、**$xxx** の値はコードの **$1** を置換するために使用されます。

コード内で参照されているパラメーターでは、スケジューリング中の解析値を追加する必要があります。

注 :

Shell ノードでは、パラメーター数が **10** に到達する場合、**${10}** を使って変数を宣言する必要があります。

変数値は、固定値です。

SQL ノードを例に説明します。 コードの **${variable name}** では、ノードのパラメーター項目 **"変数名="fixed value""** に設定します。

コード**: select xxxxxx type='${type}'**

スケジューリング変数に割り当てられる値**: type="aaa"**

スケジューリング中では、コードの変数は、**type='aaa'** によって置換されます。

変数値は、ビルトインスケジューリングパラメーターです。

SQL ノードを例に説明します。 コードの **${variable name}** では、パラメーター項目 変数名 = ノードのスケジューリングパラメーターとして設定します。

コード**: select xxxxxx dt=${datetime}**

スケジューリング変数に割り当てられる値**: datetime=$bizdate**

スケジューリング中、今日が **2017** 年 **7** 月 **22** 日の場合、コードの変数は、**dt=20170721** に置換されます。

ビルトインスケジューリングパラメーターの一覧

$bizdate: 形式 **yyyymmdd** の営業日 注記**:** このパラメーターは広く使用され、ルーチンスケジューリングでは、デフォルトで前日となります。

たとえば、**ODPS SQL** ノードのコードでは、**pt=${datetime}** です。 ノードのパラメーター設定では、**datetime=$bizdate** です。 今日が **2017** 年 **7** 月 **22** 日とします。 ノードが本日実行される場合、**$bizdate** は **pt=20170721** に置換されます。

たとえば、**ODPS SQL** ノードのコードでは、**pt=${datetime}** です。 ノードのパラメーター設定では、**datetime=$gmtdate** です。 今日が **2017** 年 **7** 月 **22** 日とします。 ノードが本日実行される場合は、**$gmtdate** は **pt=20170722** に置換されます。

たとえば、**ODPS SQL** ノードのコードでは、**pt=${datetime}** です。 ノードのパラメーター設定では、**datetime=$gmtdate** です。 今日が **2017** 年 **7** 月 **1** 日とします。 ノードが本日実行される場合は、**$bizdate** は **pt=20130630** に置換されます。

たとえば、**ODPS SQL** ノードのコードでは、**pt=${datetime}** です。 ノードのパラメーター設定では、**datetime=$gmtdate** です。 今日が **2017** 年 **7** 月 **1** 日とします。 ノードが本日実行される場合は、**$gmtdate** は **pt=20170701** に置換されます。

$cyctime: タスクのスケジュールされた時間です。 毎日のタスクでスケジュール時間が設定されていない場合、**cyctime** は現在の日の **00:00** です。 時間は、時間、分、秒に対して正確です。通常は時間単位、分単位のスケジューリングタスクに使用されます。 例**: cyctime=$cyctime**

注 **:**

$[] と **${}** を使って設定された時間パラメーターの違いについてご注意ください。 **$bizdate:** デフォルトでは、現在の日の前日の営業日です。 **$cyctime:** タスクのスケジュールされた時間です。 毎日のタスクでスケジュール時間が設定されていない場合、タスクは現在の日の **00:00** に実行されます。 時間は、時間、分、秒に対して正確です。通常は時間単位、分単位のスケジューリングタスクに使用されます。 たとえば、タスクが今日の **00:30** に実行されるようにスケジュールされている場合、スケジュール時間は、**yyyy-mm-dd 00:30:00** です。 **[]** を使って時間パラメーターが設定されている場合、**cyctime** は実行のベンチマークとして使用されます。 使用に関する詳細は、次の手順をご参照ください。 時間の算出方法は **Oracle** の算出方法と同じです。 データ作成時、置換後のパラメーター値は、営業日 **+ 1** 日です。 たとえば、日付が **20140510** を営業日として選択している場合、**cyctime** は **20140511** に置換されます。

$jobid: タスクが属するワークフローの **ID** です。 例**: jobid=$jobid**

$nodeid: ノードの **ID** です。 例**: nodeid=$nodeid**

$taskid: タスクの **ID** です。つまりノードインスタンスの **ID** です。 例**: taskid=$taskid**

$bizmonth: 形式 **yyyymm** の営業月です。

- 営業月が現在の月と同じ場合、**$bizmonth** = 営業月 - **1** です。そうでない場合は、**$bizmonth** = 営業日の月です。
- たとえば、**ODPS SQL** ノードのコードでは、**pt=${datetime}** です。 ノードのパラメーター設定では、**datetime=$bizmonth** です。 今日が **2017** 年 **7** 月 **22** 日とします。 ノードが今日実行される場合、**$bizmonth** は **pt=201706** に置換されます。

$gmtdate: 形式 **yyyymmdd** の現在の日です。 このパラメーターの値は、デフォルトでは今日です。 データ作成時、入力した **gmtdate** は営業日 プラス **1** です。

カスタムパラメーター **${…}** パラメーターの説明**:**

- **$bizdate** に基づいてカスタマイズされた時間形式では、**yyyy** は **4** 桁の年、**yy** は **2** 桁の月、**mm** は月、**dd** は日を示します。 パラメーターは意図したとおりに組み合わせることができます。たとえば、**${yyyy}**、 **${yyyymm}**、 **${yyyymmdd}**、 **${yyyy-mm-dd}** です。
- **$bizdate** は年、月、日に対して正確です。 したがって、カスタムパラメーター **${……}** は、年、月、日のみを表現できます。
- 特定の期間のプラス、マイナスの期間を取得する方法**:**

  次の **N** 年**: ${yyyy+N}**

  前の **N** 年**: ${yyyy-N}**

  次の **N** 月**: ${yyyymm+N}**

  前の **N** 月**: ${yyyymm-N}**

  次の **N** 週間**: ${yyyymmdd+7*N}**

  前の **N** 週間**: ${yyyymmdd-7*N}**

  次の **N** 日**: ${yyyymmdd+N}**

  前の **N** 日**: ${yyyymmdd-N}**

${yyyymmdd}:** 形式 **yyyymmdd** の営業日です。 **$bizdate の値と一致します。

- 注記**: $bizdate** の値と一致します。 このパラメーターは広く使用されます。ルーチンスケジューリングではデフォルトで、前日の日付となります。 このパラメーターの形式をカスタマイズすることができます。たとえば、**${yyyy-mm-dd}** の形式を **yyyy-mm-dd** にします。
- たとえば、**ODPS SQL** ノードのコードでは、**pt=${datetime}** です。 ノードのパラメーター設定は、**datetime=${yyyymmdd}** です。 今日が **2013** 年 **7** 月 **22** 日とします。 ノードが今日実行される場合、**${yyyymmdd}** は **pt=20130721** に置換されます。

${yyyymmdd-/+N}: yyyymmdd プラス、またはマイナス **N** 日

${yyyymm-/+N}: yyyymm プラス、またはマイナス **N** 月

${yyyy-/+N}: 年 **(yyyy)** プラス、またはマイナス **N** 年

${yy-/+N}: 年 **(yy)** プラス、またはマイナス **N** 年

注記**: yyyymmdd** は営業日を示し、**yyyy-mm-dd** のように区切りをサポートします。 上記のパラメーターは、営業日の年、月、日に由来しています。

例**:**

- **ODPS SQL** ノードのコードでは、**pt=${datetime}** です。 ノードのパラメーター設定では、**datetime=${yyyy-mm-dd}** です。 今日が **2018** 年 **7** 月 **22** 日とします。 ノードが今日実行される場合、**${yyyy-mm-dd}** は **pt=2018-07-21** に置換されます。
- **ODPS SQL** ノードのコードでは、**pt=${datetime}** です。 ノードのパラメーター設定では、**datetime=${yyyymmdd-2}** です。 今日が **2018** 年 **7** 月 **22** 日とします。 ノードが今日実行される場合、**${yyyymmdd-2}** は **pt=20180719** に置換されます。
- **ODPS SQL** ノードのコードでは、**pt=${datetime}** です。 ノードのパラメーター設定では、**datetime=${yyyymm-2}** です。 今日が **2018** 年 **7** 月 **22** 日とします。 ノードが今日実行される場合、**${yyyymm-2}** は **pt=201805** に置換されます。
- **ODPS SQL** ノードのコードでは、**pt=${datetime}** です。 ノードのパラメーター設定では、**datetime=${yyyy-2}** です。 今日が **2018** 年 **7** 月 **22** 日とします。 ノードが今日実行される場合、**${yyyy-2}** は **pt=2018** に置換されます。

ODPS SQL ノード設定では、複数のパラメーターに値が割り当てられます。たとえば、**startdatetime=$bizdate enddatetime=${yyyymmdd+1} starttime=${yyyy-mm-dd} endtime=${yyyy-mm-dd+1}** です。

例**: ($cyctime=20140515103000** と想定します**)**

- **$[yyyy] = 2014, $[yy] = 14**、 **$[mm] = 05**、 **$[dd] = 15**、 **$[yyyy-mm-dd] = 2014-05-15**、 **$[hh24:mi:ss] = 10:30:00**、 **$[yyyy-mm-dd hh24:mi:ss] = 2014-05-1510:30:00**
- **$[hh24:mi:ss - 1/24] = 09:30:00**
- **$[yyyy-mm-dd hh24:mi:ss -1/24/60] = 2014-05-1510:29:00**
- **$[yyyy-mm-dd hh24:mi:ss -1/24] = 2014-05-1509:30:00**
- **$[add_months(yyyymmdd,-1)] = 2014-04-15**
- **$[add_months(yyyymmdd,-12*1)] = 2013-05-15**
- **$[hh24] =10**
- **$[mi] =30**

パラメーター **$cyctime** をテストする方法

インスタンスが実行された後、ノードを右クリックしノードの属性を確認します。 スケジュールされた時間がインスタンスが周期的に実行される時間かどうかを確認します。

パラメーター値がスケジュールした時間マイナス 1 時間に置換された後の結果です。

1.10 コンポーネントの管理

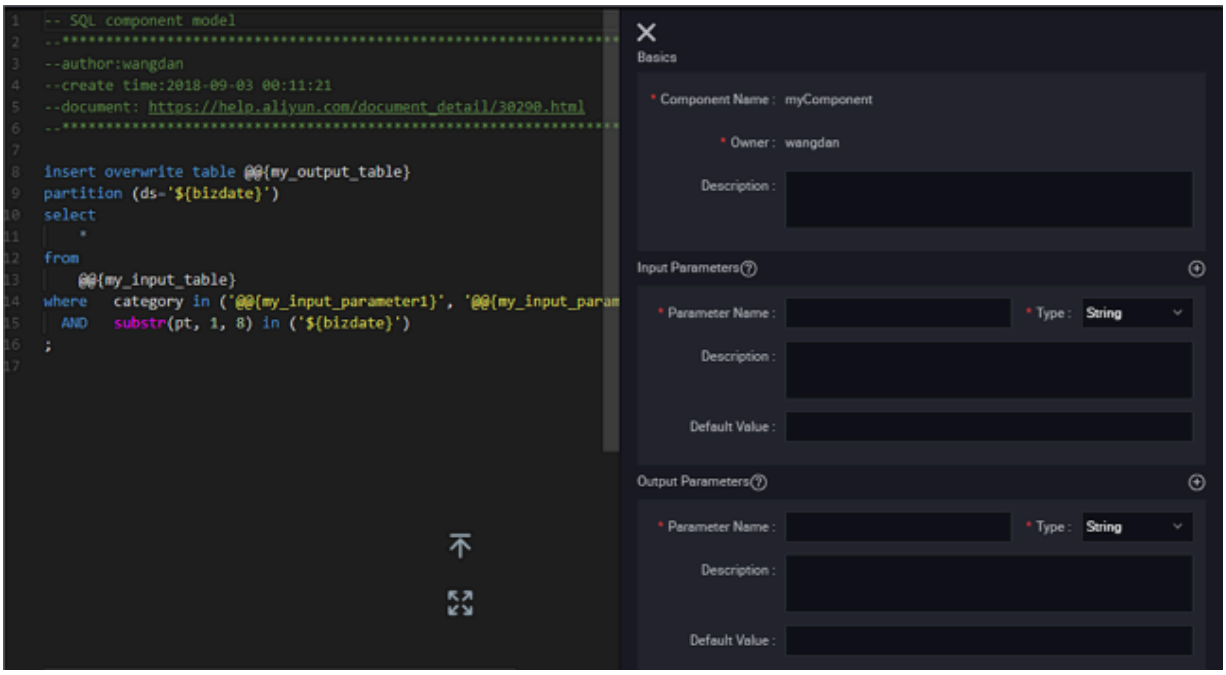
1.10.1 コンポーネントの使用

開発効率を向上させるため、データタスク開発者はプロジェクトとテナントから提供されるコンポーネントを使って、データプロセスノードを作成します。

- ローカルプロジェクトのメンバーが作成したコンポーネントは **[Project Components]** の下にあります。
- テナントメンバーが作成したコンポーネントは **[Public Components]** の下にあります。

コンポーネントの使用方法については、「*SQL コンポーネントノード*」をご参照ください。

インターフェイスの機能



インターフェイスの機能は次のとおりです。

No.	機能	説明
1	**Save**	クリックして、現在のコンポーネントの設定を保存します。

No.	機能	説明
2	**Steallock Edit**	現在のコンポーネントの所有者ではない場合、クリックし、ノードを **steallock** 編集します。
3	**Submit**	クリックして、現在のコンポーネントを開発環境へ送信します。
4	**Publish Component**	クリックして、ユニバーサルグローバルコンポーネントをテナント全体へ発行します。テナントのすべてのユーザーがパブリックコンポーネントを表示し使用することができます。
5	**Resolve Input and Output Parameters**	クリックして、現在のコードの入出力パラメーターを解決します。
6	**Pre-compile**	クリックして、現在のコンポーネントのカスタムおよびコンポーネントパラメーターを編集します。
7	**Run**	クリックして、開発環境でコンポーネントをローカルで実行します。
8	**Stop Run**	クリックして、実行中のコンポーネントを停止します。
9	**Format**	クリックして、現在のコンポーネントをキーワード別にソートします。
10	**Parameter settings**	クリックして、コンポーネント情報、入力パラメーターの設定、出力パラメーターの設定を表示します。
11	**Version**	クリックして、現在のコンポーネントの送信および解放履歴を表示します。
12	**Reference Records**	クリックして、コンポーネントの使用履歴を表示します。

1.10.2 コンポーネントの作成

コンポーネントの定義

コンポーネントは、複数の入出力パラメーターを含む **SQL** コードプロセステンプレートです。**SQL** コードプロセスを処理するため、**1** つ以上のソースデータテーブルがインポート、フィルタ、結合、および集約され、新しいビジネスで必要とされるターゲットテーブルを形成します。

コンポーネントの値

実際のビジネスでは、**SQL** コードプロセスは類似しています。 処理上にある入力と出力テーブルは、同じまたは互換性のある構造になっていますが、名前が異なります。 この場合、コンポーネントの開発者は、このような **SQL** プロセスを **SQL** コンポーネントノードへ、**SQL** プロセスにある変数入出力テーブルを入出力パラメーターへ要約し、**SQL** コードを再利用します。

SQL コンポーネントノードを使用する際、コンポーネントユーザーはコンポーネントの一覧からビジネスフローを選択するように、コンポーネントを選択し、これらコンポーネント向けのビジネスにある特定の入出力テーブルを設定します。コードのコピーを繰り返し行わずに新しい SQL コンポーネントノードを生成します。 これは、開発効率を大幅に向上させ、繰り返し開発を行うことを防ぐことができます。 生成後のSQL コンポーネントノードの発行とスケジューリングは、共通 SQL ノードの発行とスケジューリングと同じです。

コンポーネントの構成

関数の定義と同じように、コンポーネントは入力パラメーター、出力パラメーター、コンポーネントコードプロセスによって構成されています。

コンポーネント入力パラメーター

コンポーネント入力パラメーターには、名前、タイプ、説明、定義といった属性が含まれます。パラメーターのタイプは、テーブルまたは文字列です。

- テーブルタイプのパラメーターは、コンポーネントプロセスで参照されるテーブルを指定します。 コンポーネントを使用する際、コンポーネントユーザーは特定のビジネスで必要とされるテーブルにパラメーターを設定します。
- 文字列タイプのパラメーターは、コンポーネントプロセスでの変数制御パラメーターを指定します。 たとえば、特定のプロセスの結果テーブルが各リージョンの トップ N 都市の販売額のみを出力する場合、値 N は文字列パラメーターで指定することができます。

  特定のプロセスの結果テーブルがその省の販売額の総額を出力する必要がある場合、省文字列タイプのパラメーターを設定し、異なる省を指定し、その指定した省の販売額を取得します。
- パラメーターの説明は、コンポーネントプロセスのパラメーターの役割を指定します。
- パラメーターの定義は、テーブル構造のテキスト定義で、これはテーブルタイプのパラメーターに対してのみ要求されます。 この属性を指定する場合、コンポーネントプロセスが適切に実行されるように、コンポーネントユーザーはテーブルパラメーターで定義されているフィールド名とタイプに互換性のある入力テーブルを提供する必要があります。 そうでない場合、入力テーブル内に指定されたフィールドが見つからないため、コンポーネントプロセスが実行される際にエラーが報告されます。 入力テーブルには、テーブルパラメーターで定義されたフィールド名とタイプが含まれている必要があります。 フィールドとタイプの順序は異なっていても、また入力テーブルに他のフィールドが含まれていてもかまいません。 定義は、参照目的のみです。 ユーザーに対してガイダンスを提供し、すぐに強制的に確認する必要はありません。
- 推奨されるテーブルパラメーターの定義形式は次のとおりです。

```
Field 1 name Field 1 type Field 1 comment
```

```
Field 2 name Field 2 type Field 2 comment
Field n name Field n type Field n comment
```

例:

```
area_id string ‘Region ID’
city_id string ‘City ID’
order_amt double ‘Order amount’
```

コンポーネント出力パラメーター

- コンポーネント出力パラメーターには、名前、タイプ、説明、定義といった属性が含まれます。 パラメータータイプは、テーブルのみです。 文字列出力パラメーターは論理的な意味を持ちません。
- テーブルタイプパラメーター: コンポーネントプロセスから生成されるテーブルを指定します。 コンポーネントを使用する際、コンポーネントユーザーは、特定のビジネスに対してコンポーネントプロセスが生成する結果テーブルに対してパラメーターを設定します。
- パラメーターの説明: コンポーネントプロセスのパラメーターの役割を指定します。
- パラメーターの定義: テーブル構造のテキスト定義です。 この属性を指定する場合、コンポーネントプロセスが適切に実行されるように、コンポーネントユーザーはテーブルパラメーターで定義されているフィールド名とタイプの数と同じ数を持つ出力テーブルとパラメーターを提供する必要があります。 そうでない場合、フィールド数が一致しない、またはタイプに互換性がないため、コンポーネントプロセスが実行される際エラーが報告されます。 出力テーブルのフィールド名は、テーブルパラメーターで定義された名前とは一致しません。 定義は参照目的のみです。 ユーザーに対してガイダンスを提供し、すぐに強制的に確認する必要はありません。
- 推奨されるテーブルパラメーターの定義形式は次のとおりです。

```
Field 1 name Field 1 type Field 1 comment
Field 2 name Field 2 type Field 2 comment
Field n name Field n type Field n comment
```

例:

```
area_id string ‘Region ID’
city_id string ‘City ID’
order_amt double ‘Order amount’
rank bigint ‘Rank’
```

コンポーネントプロセス体

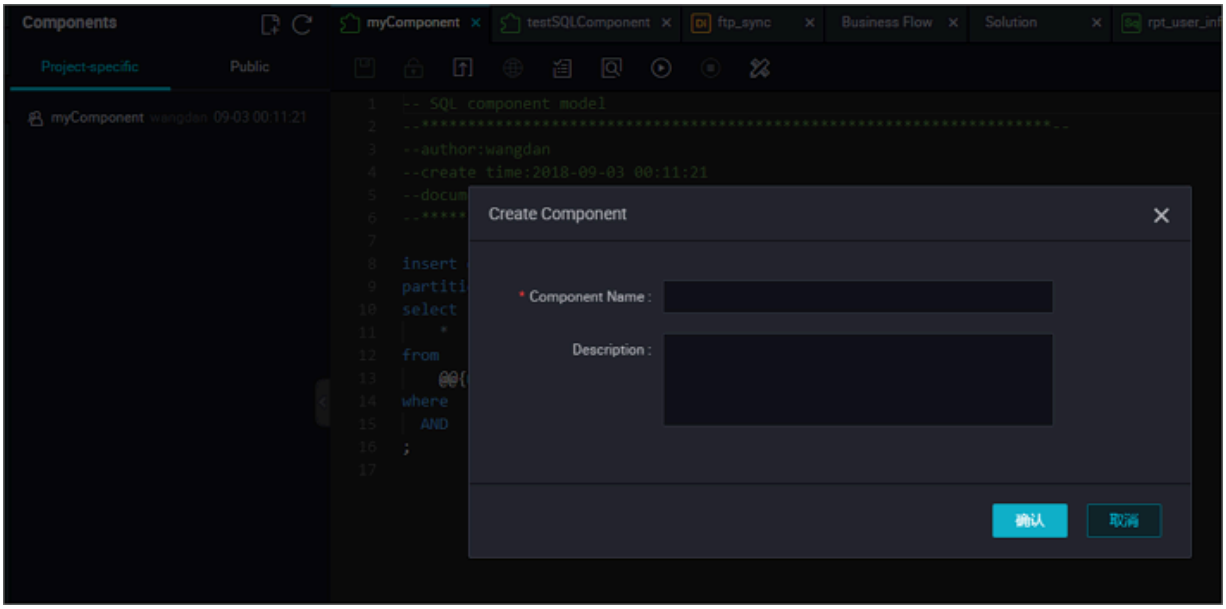
プロセス体のパラメーターの参照形式は、 **@@{parameter name}**です。

要約 **SQL** ワーキングプロセスのコンパイルによって、プロセス体は入力パラメーターに基づいて指定された入力テーブルを制御し、ビジネス値を持つ出力テーブルを生成します。

コンポーネントプロセスの開発には特定のスキルが必要となります。 入出力パラメーターの異なる値が正しく実行可能な SQL コードを生成できるように、入出力パラメーターはコンポーネントプロセスコードで良く使用されます。

コンポーネントの作成例

次の図に示すとおり、コンポーネントを作成します。



ソーステーブルスキーマの定義

営業データのソース MySQL スキーマの定義は次の表で説明するとおりです。

フィールド名	フィールドタイプ	フィールドの説明
order_id	**varchar**	注文 ID
report_date	**datetime**	注文日
customer_name	**varchar**	顧客名
order_level	**varchar**	注文グレード
order_number	**double**	注文数
order_amt	**double**	注文金額
back_point	**double**	割引
shipping_type	**varchar**	輸送モード
profit_amt	**double**	利益額
price	**double**	単価
shipping_cost	**double**	輸送コスト
area	**varchar**	リージョン

フィールド名	フィールドタイプ	フィールドの説明
province	**varchar**	省
city	**varchar**	市
product_type	**varchar**	プロダクトタイプ
product_sub_type	**varchar**	プロダクトサブタイプ
product_name	**varchar**	プロダクト名
product_box	**varchar**	プロダクト梱包箱
shipping_date	**Datetime**	輸送日

コンポーネントのビジネス的な意味合い

コンポーネント名**: get_top_n**

コンポーネントの説明**:**

コンポーネントプロセスでは、指定した営業データテーブルを入力パラメーター (テーブルタイプ) として使用し、トップ都市の数を入力パラメーター (文字列タイプ) として使用します。都市は販売金額別にランク付けされます。 このように、コンポーネントユーザーは各リージョンの指定したトップ **N** 都市のランキングを簡単に取得することができます。

コンポーネントパラメーターの定義

入力パラメーター **1:**

パラメーター名**: myinputtable type: table**

入力パラメーター **2**

パラメーター名**: topn type: string**

入力パラメーター **3**

パラメーター名**: myoutput type: table**

パラメーターの定義

area_id string

city_id string

order_amt double

rank bigint

テーブル作成文:

```
CREATE TABLE IF NOT EXISTS company_sales_top_n
(
area STRING COMMENT 'Region',
city STRING COMMENT 'City',
sales_amount DOUBLE COMMENT 'Sales amount',
rank BIGINT COMMENT 'Rank'
)
COMMENT 'Company sales ranking'
PARTITIONED BY (pt STRING COMMENT '')
LIFECYCLE 365;
```

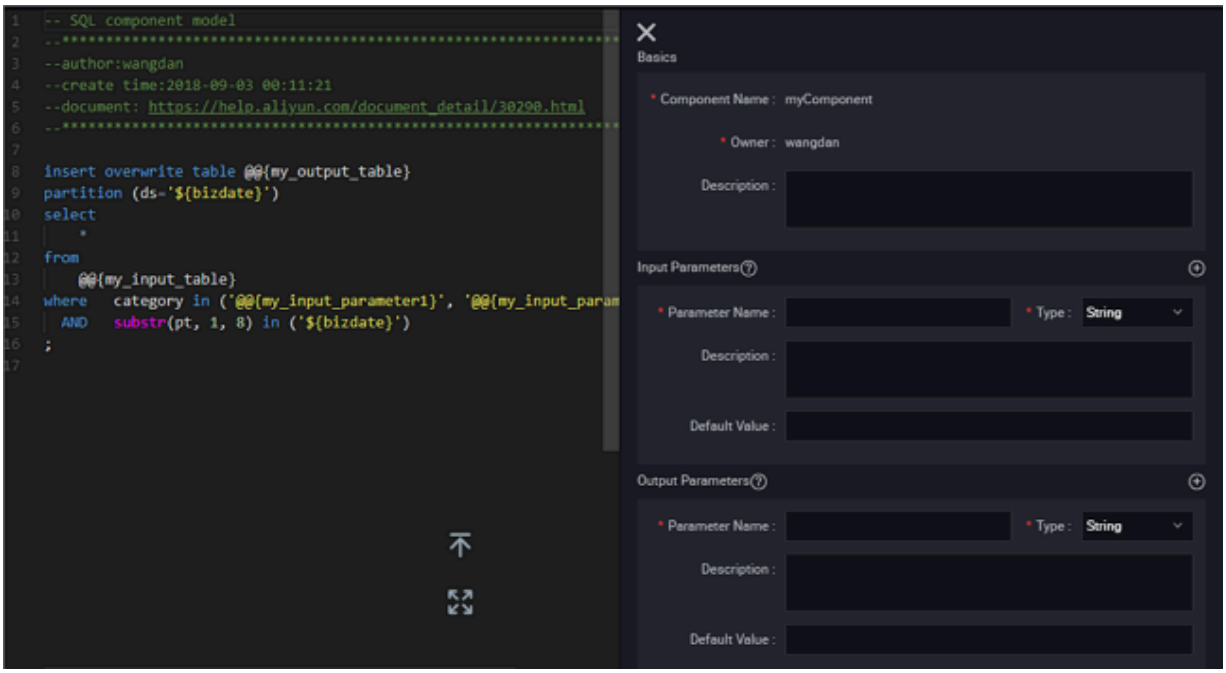
コンポーネントプロセス体の定義

```
INSERT OVERWRITE TABLE @@{myoutput} PARTITION (pt='${bizdate}')
    SELECT r3.area_id,
    r3.city_id,
    r3.order_amt,
    r3.rank
from (
SELECT
    area_id,
    city_id,
    rank,
    order_amt_1505468133993_sum as order_amt ,
    order_number_1505468133991_sum,
    profit_amt_1505468134000_sum
FROM
    (SELECT
    area_id,
    city_id,
    ROW_NUMBER() OVER (PARTITION BY r1.area_id ORDER BY r1.order_amt_
1505468133993_sum DESC)
AS rank,
    order_amt_1505468133993_sum,
    order_number_1505468133991_sum,
    profit_amt_1505468134000_sum
FROM
    (SELECT area AS area_id,
     city AS city_id,
     SUM(order_amt) AS order_amt_1505468133993_sum,
     SUM(order_number) AS order_number_1505468133991_sum,
     SUM(profit_amt) AS profit_amt_1505468134000_sum
FROM
    @@{myinputtable}
WHERE
    SUBSTR(pt, 1, 8) IN ( '${bizdate}' )
GROUP BY
    area,
    city )
    r1 ) r2
WHERE
    r2.rank >= 1 AND r2.rank <= @@{topn}
ORDER BY
    area_id,
    rank limit 10000) r3;
```

コンポーネントの共有スコープ

共有スコープは、プロジェクトコンポーネントとパブリックコンポーネントの **2** つがあります。

コンポーネントを発行した後、デフォルトでプロジェクト内に表示されます。 コンポーネント開発者は、**[Publish Component]** アイコンをクリックし、ユニバーサルグローバルコンポーネントを全体のテナントへ発行します。テナント内のすべてのユーザーがパブリックコンポーネントを表示し使用することができます。 コンポーネントがパブリックかどうかは、次の図のアイコンが表示されているかどうかによります。



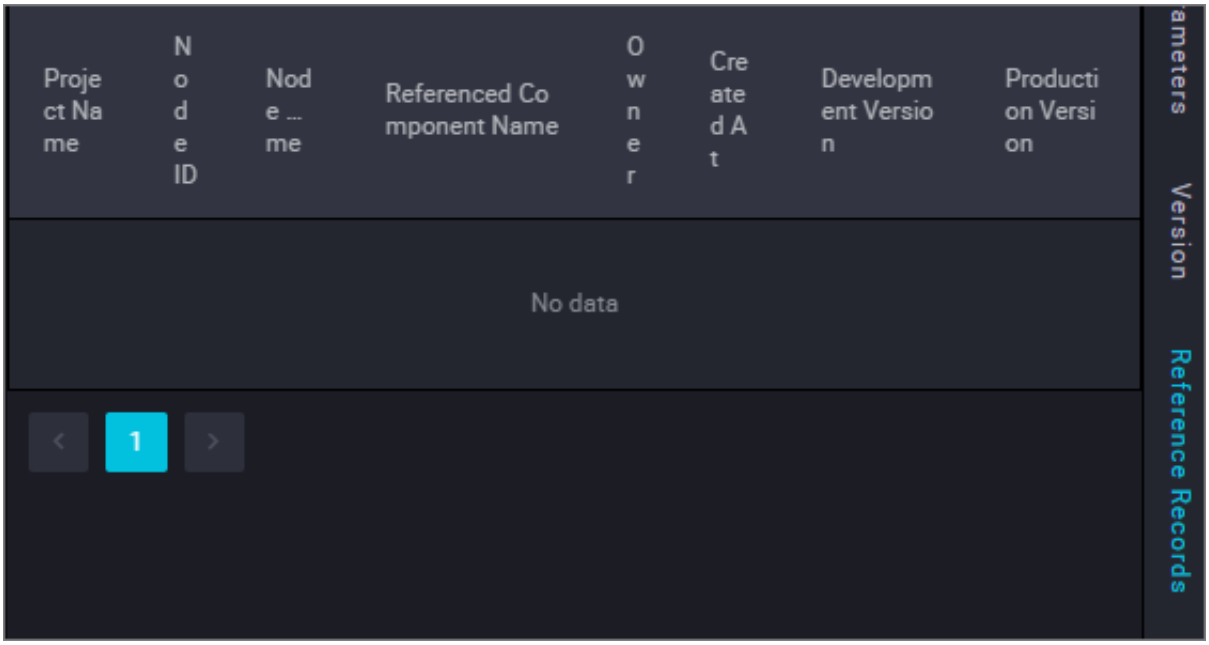
コンポーネントの使用

開発されたコンポーネントはどのように使用しますか。 詳細については、次をご参照ください。

コンポーネントの使用

コンポーネントの参照履歴

コンポーネント開発者は、**[Reference Records]** タブをクリックし、コンポーネントの参照履歴を表示します。

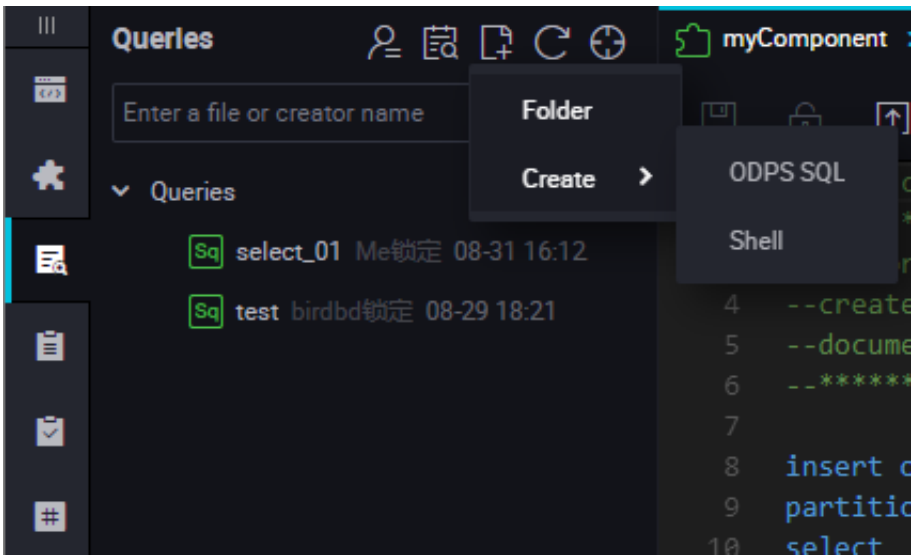


1.11 クエリ

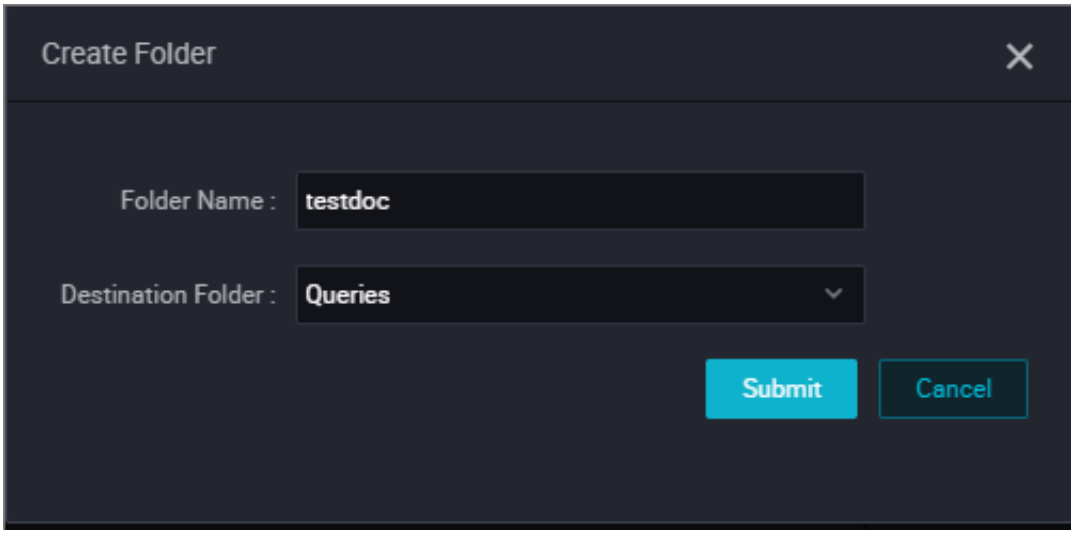
一時的なクエリは、コード編集の使用を容易にし、ローカルコードの実際の状態が想定通りになっているかどうかをテストし、コードステータスを確認します。 したがって、一時的なクエリは送信、解放、スケジューリングパラメーターの設定をサポートしません。 スケジューリングパラメーターを使用するには、**[Data development]** または **[Manual business flow]** でノードを作成します。

フォルダの作成

1. 左側のナビゲーションウィンドウで、**[Queries]** をクリックし、**[folder]** を選択します。



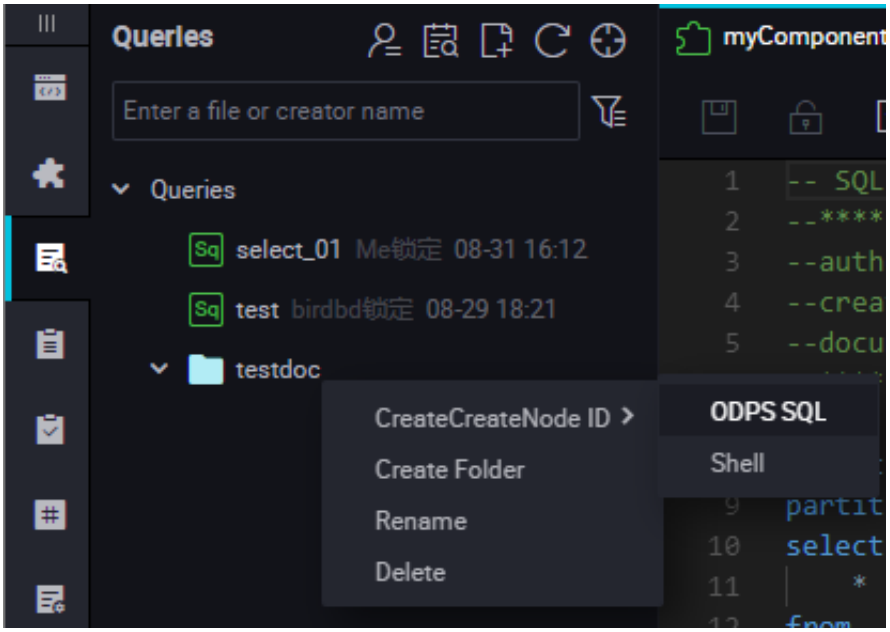
2. フォルダ名を入力し、フォルダディレクトリを選択します。**[Submit]** をクリックします。



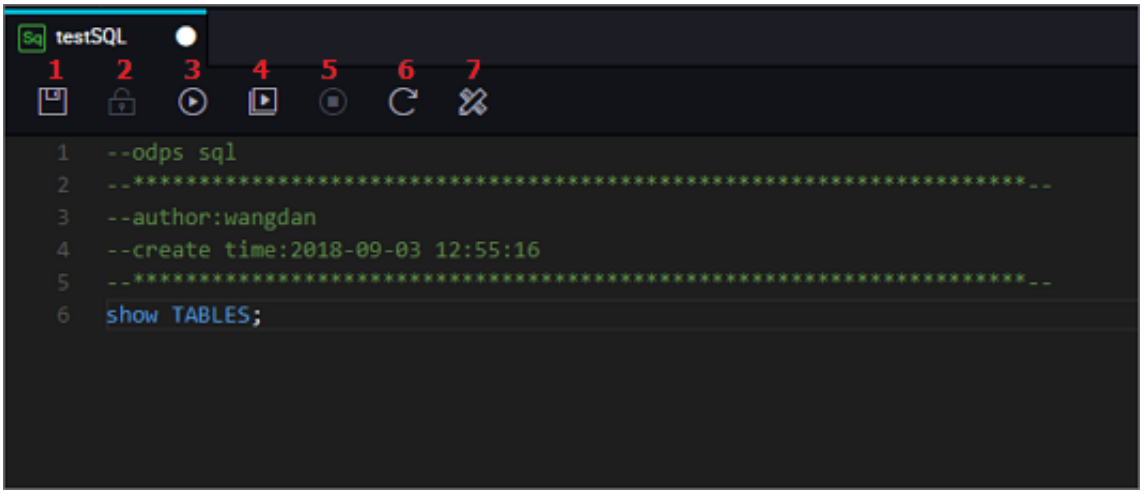
> 注：
> マルチレベルのフォルダディレクトリがサポートされています。 したがって、作成した別のフォルダにフォルダを格納することができます。

ノードの作成

一時的なクエリは、**SHELL** および **SQL** ノードのみをサポートします。



ODPS SQL ノードを例に説明します。フォルダ名を右クリックし、**[Create Node]** > **[ODPS SQL]** を選択します。



No.	機能	説明
1	**Save**	クリックして、入力したコードを保存します。
2	**Steallock Edit**	ノードの所有者ではないユーザーがクリックして、ノードを編集します。
3	**Run**	クリックしてコードをローカルで (開発環境内で) 実行します 。
4	**Advanced Run (with Parameters)**	クリックして、コードに設定されたパラメーターを使って、現在のノードのコードを実行します。 注： **Advanced Run** は **Shell** ノードでは利用できません。
5	**Stop Run**	クリックして、実行中のコードを停止します。
6	**Reload**	クリックして、ページを更新し、リロードし、最後に保存された状態を復元します。 保存されていないコンテンツは失われます。 注： 設定センターでキャッシュを有効にしている場合、ページを更新した後、保存されていないコードがキャッシュされたことを示すメッセージが表示されます。 必要なバージョンを選択します。
7	**Format**	クリックして、現在のノードコードをキーワード形式でソートします。 コード行が長すぎるときに使用します。

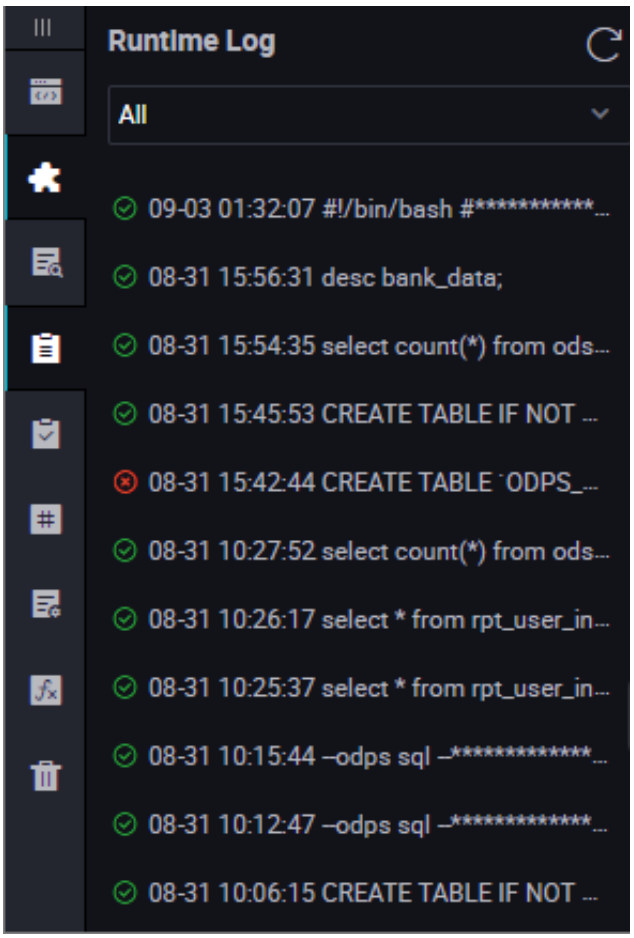
1.12 ランニングログ

ランニングログ (**Running Log**) ページは、過去 **3** 日間にローカルで実行されたすべてのタスクの履歴を表示します。 クリックして、タスクの履歴を表示し、タスクステータス別に実行履歴をフィルタします。

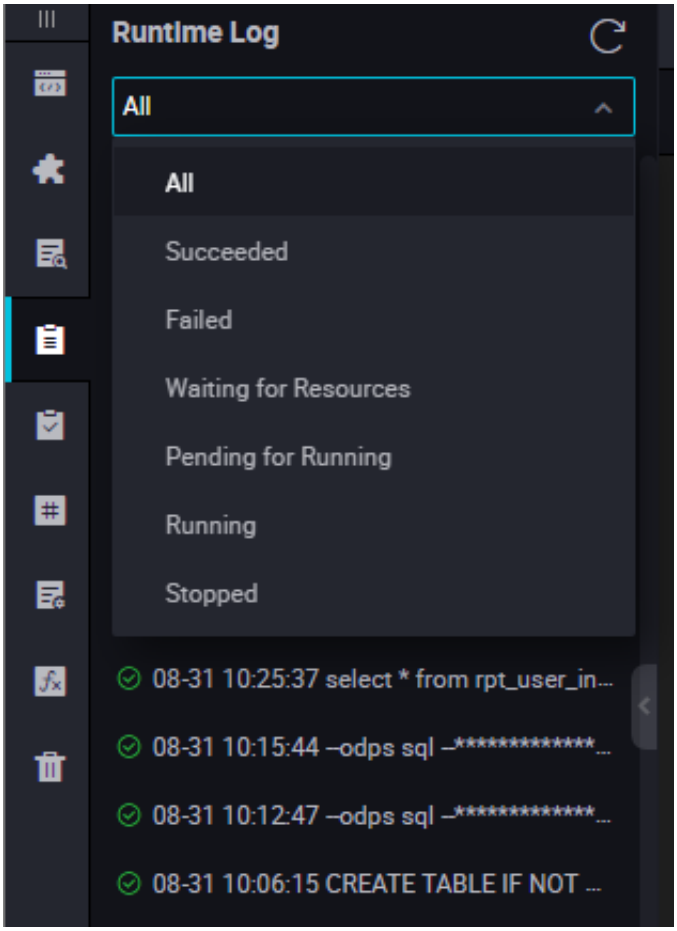
> 注：
> ランニングログは、**3** 日分のみ取得できます。

ランニングログの表示

1. クリックして、**[Running Log]** ページへ移動します (デフォルトでは、すべてのステータスのタスクが表示されます)。



2. ドロップダウンリストボックスをクリックし、タスクフィルタ条件を選択します。



3. ターゲットランニング履歴をクリックします。 ランニングログページでは、ランニング履歴のログが表示されます。

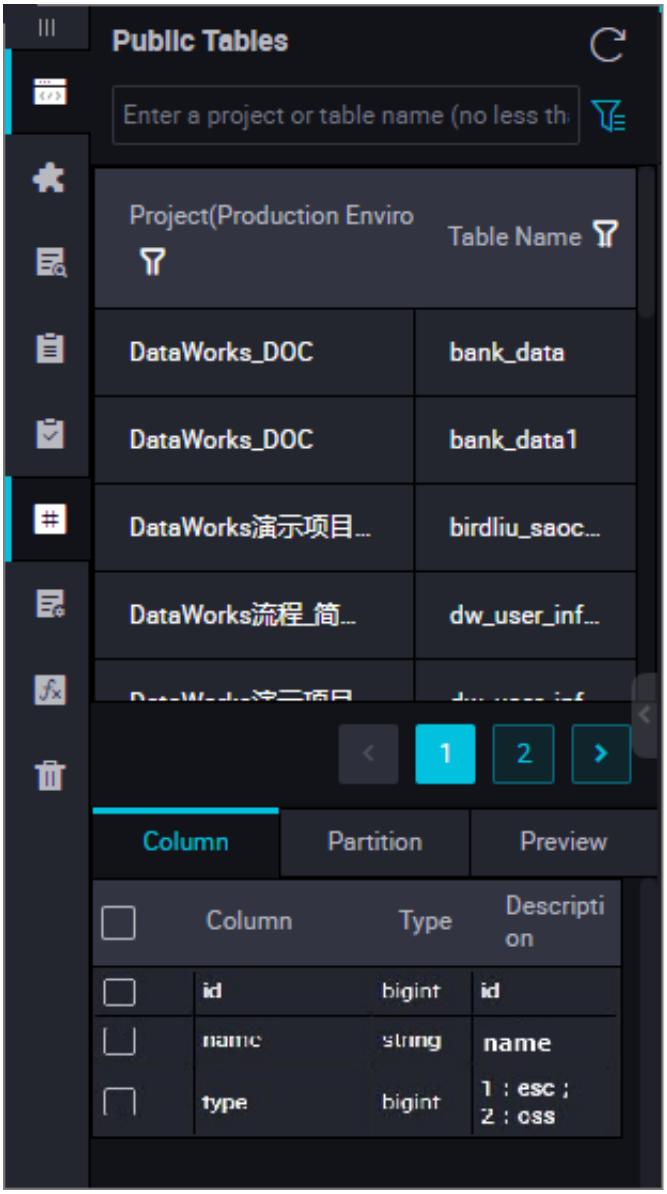
ログを一時的ファイルへ保存します。

ランニング履歴の **SQL** 文を保存するには、**[Save]** アイコンをクリックして、一時的ファイルに実行された **SQL** 文を保存します。

ファイル名とディレクトリを入力し、**[Submit]** をクリックします。

1.13 パブリックテーブル (Public Table)

[Public Table] エリアでは、現在のテナントにあるすべてのプロジェクトで作成されたテーブルを表示できます。



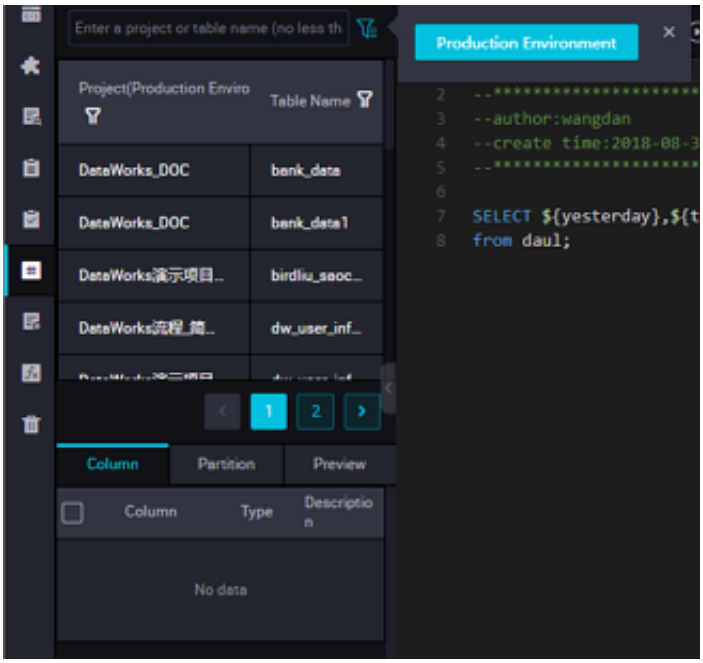
- **Project:** プロジェクト名 プレフィックス **"odps."** が、各プロジェクト名に追加されています。 たとえば、プロジェクト名が **test** の場合、**"odps.test"** と表示されます。
- **Table Name:** プロジェクトのテーブル名です。

テーブル名をクリックして、テーブルの列やパーティション情報を表示し、テーブルデータをプレビューします。

- **Column Information:** クリックして、テーブルのフィールド数、フィールドタイプ、フィールドの説明を表示します。
- **Partition Information:** クリックして、テーブルのパーティション情報とパーティション数を表示します。 最大 **6,000** 個のパーティションが許可されています。 ライフサイクルを設定した場合、パーティションの実際の数は、ライフサイクルに依存します。
- **Data Preview:** クリックして、現在のテーブルのデータをプレビューします。

環境切り替え

Table Management 同様に、**Public Table** は開発環境と運用環境をサポートしています。 現在の環境は青で表示されます。 照会する環境をクリックすると、対応する環境が表示されます。

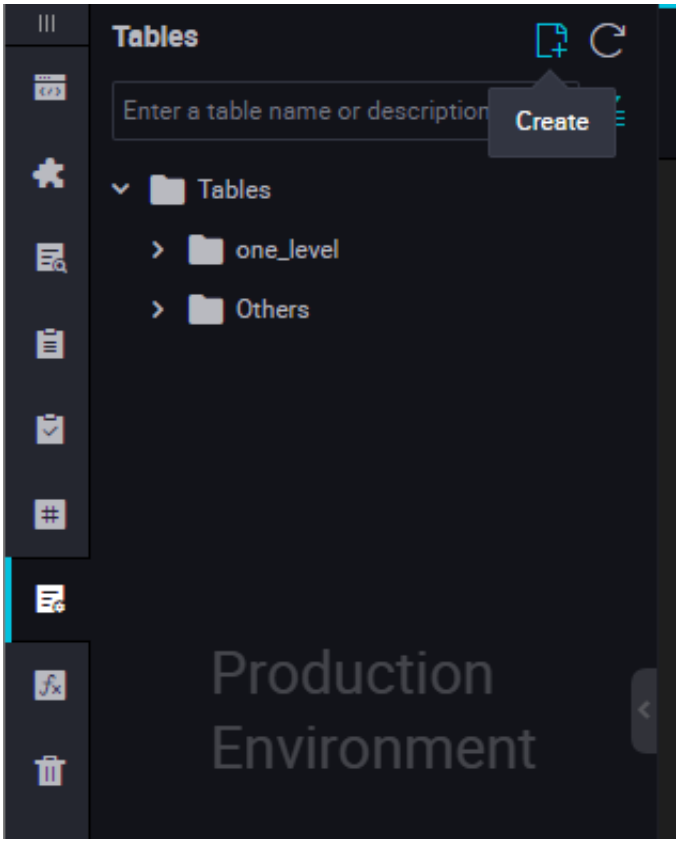


1.14 テーブルの管理

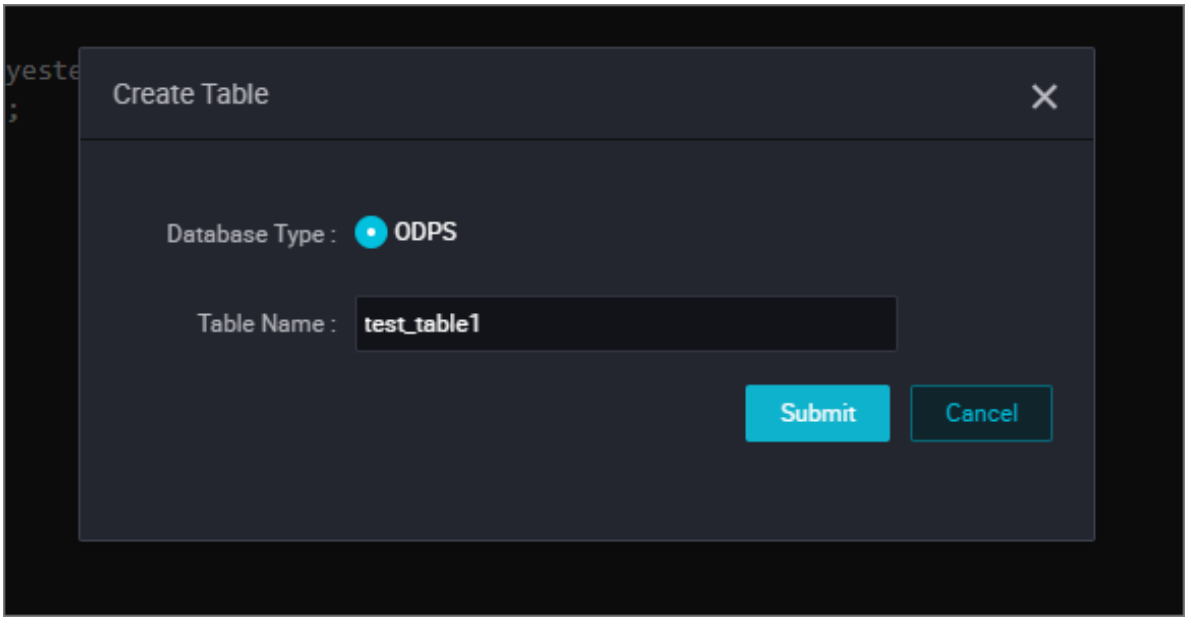
テーブルの作成

1. ページの左上隅にある **[Table Management]** をクリックします。

2. [+] アイコンを選択し、テーブルを作成します。

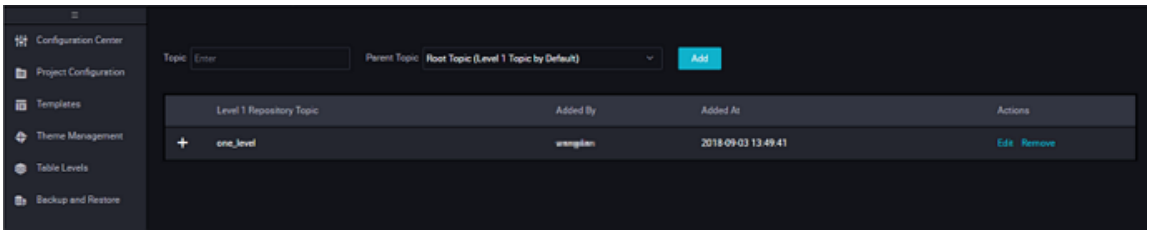


3. テーブル名を入力します。**MaxCompute** テーブルのみが現在サポートされています。**[Submit]** をクリックします。



4. 基本的な属性を設定します。

   - **Chinese Name:** 作成するテーブルの中国語名です。
   - **Level-1 Topic:** 作成するテーブルの **level-1** ターゲットフォルダの名前です。
   - **Level-2 Topic:** 作成するテーブルの **level-2** ターゲットフォルダの名前です。
   - **Description:** 作成するテーブルの説明です。
   - **[Create Topic]** をクリックします。 表示された **[Topic Management]** ページで、**level-1** と **level-2** トピックを作成します。



5. **DDL** モードでテーブルを作成します。

   [DDL Mode] をクリックします。 表示されたダイアログボックスで、標準テーブル作成文を入力します。

   テーブル作成 **SQL** 文を編集した後、**[Generate Table Structure]** をクリックします。 **[Basic Attributes]**、**[Physical Model Design]**、**[Table Structure Design]** エリアの情報は自動的に入力されます。

6. **GUI** でテーブルを作成します。

   DDL モードでのテーブルの作成が適切ではない場合、**GUI** で次の設定を行ってテーブルを作成します。

   - **Physical model design**
     - **Table type: [Partitioned Table]** または **[Non-partitioned Table]** を設定します。
     - **Life Cycle: MaxCompute** のライフサイクル機能です。 **Life Cycle** で指定した周期 (単位: 日) でアップデートされていないテーブル (またはパーティション) のデータはクリアされます。
     - **Level: [DW]**、**[ODS]**、または **[RPT]** を設定します。
     - **Physical Category: [Basic Business Layer]**、**[Advanced Business Layer]**、または **[Other]** を設定します。 **[Create Level]** をクリックします。 表示された **[Level Management]** ページで、レベルを作成します。
   - **Table structure design**
     - **English Field Name:** フィールドの英語名で、文字、数字、アンダースコア (_) を含めることができます。
     - **Chinese Name:** フィールドの中国語の略名です。
     - **Field Type: MaxCompute** データタイプで、**String**、**Bigint**、**Double**、**Datetime**、**Boolean** 型のみです。
     - **Description:** フィールドの詳細説明です。
     - **Primary Key:** 選択し、フィールドが主キーか、結合主キーにあるフィールドかを示します。
     - **[Add Field]** をクリックし、新しいフィールドに列を追加します。
     - **[Delete Field]** をクリックし、作成したフィールドを削除します。

        注 :
       作成したテーブルからフィールドを削除し、テーブルを再度送信した場合、現在のテーブルを破棄して、同じ名前のテーブルを新しく作成する必要があります。 この操作は運用環境では行えません。
     - **[Move Up]** をクリックし、作成したテーブルのフィールドの順番を調整します。 しかし、作成したテーブルのフィールドの順番を調整するには、現在のテーブルを破棄し

て、同じ名前のテーブルを新しく作成する必要があります。 この操作は運用環境では行えません。

- **[Move Down]** をクリックします。操作は **[Move Up]** と同じです。
- **[Add Partition]** をクリックし、現在のテーブルにパーティションを作成します。 作成したテーブルにパーティションを追加するためには、現在のテーブルを破棄して、同じ名前のテーブルを新しく作成する必要があります。 この操作は運用環境では行えません。
- **[Delete Partition]** をクリックしてパーティションを削除します。 現在のテーブルからパーティションを削除するには、現在のテーブルを破棄して、同じ名前のテーブルを新しく作成する必要があります。 この操作は運用環境では行えません。
- **Action:** 新しいフィールドの送信、フィールドの削除、また他の属性を編集します。

  プロパティには、主にシステムに対して検証ロジックを生成するために提供されるデータ品質に関連する情報が含まれています。 データプロファイル、**SQL** スキャン、テストルール作成などのシナリオで使用されます。

  ■ **0 Allowed:** 選択すると、フィールド値をゼロにすることができます。 このオプションは、**bigint** と **Double** 型フィールドのみに適用されます。

  ■ **Negative Value Allowed:** 選択すると、フィールド値を負の値にすることができます。 このオプションは、**bigint** と**Double** 型フィールドのみに適用されます。

  ■ **Security Level:** セキュリティレベルは **0 - 4** です。 数字が大きくなると、セキュリティ要件も高くなります。 セキュリティレベルがデジタル要件を満たさない場合、フォームの対応するフィールドへアクセスできません。

  ■ **Unit:** 金額です。ドル、またはセントにすることができます。 このオプションは金額に関連しないフィールドには不要です。

  ■ **Lookup Table Name/Kay Value:** メンバータイプやステータスといった列挙型フィールドに適用されます。 このフィールドに対応した辞書テーブル (またはディメンションテーブル) の名前を入力します。 たとえば、メンバーステータスに対応した辞書テーブル名は、**"dim_user_status"** です。 グローバルユニークな辞書テーブル

を使用する場合、辞書テーブル内のフィールドに対応する **key_type** を入力します。たとえば、メンバーステータスの対応キー値は、**"AOBAO_USER_STATUS"** です。

- **Value Range:** 現在のフィールドの最大値と最小値です。 **Bigint** と **Double** のフィールドにのみに適用されます。
- **Regular Expression Verification:** 現在のフィールドで使用する正規表現です。たとえば、フィールドが携帯電話番号で、正規表現 (またはさらに厳格な制限) で **11** 桁の数字に制限することができます。
- **Maximum Length:** フィールド値の最大文字数です。 **String** 型のフィールドにのみ適用されます。
- **Date Precision:** 日付の精度です。**[Hour]**、**[Day]**、**[Month]**、**[others]**のいずれかに設定できます。 たとえば、フィールド値が **2014-08-01**であっても (精度は **Day** のように見えます)、月次サマリーテーブル **"month_id"** の精度は **Month** です。 **Datetime** または **String** タイプの日付値に適用されます。
- **Date Format: String** タイプの日付値にのみ適用されます。 フィールドに実際に格納される日付値の形式は、**yyyy-mm-dd hh:mm:ss** のようになります。
- **KV Primary Separator/Secondary Separator:** キー値ペアが組み合わされた大規模なフィールド **(String** 型) に適用されます。 たとえば、プロダクト拡張属性に **" key1:value1;key2:value2;key3:value3;..."** といった値がある場合、セミコロン **(;)** はフィールドの主となるセパレーターで、キー値ペアを区切ります。コロン **(:)** は二次的なセパレーターで、キー値ペアのキーと値を区切ります。

- **Partition Field Design:** このオプションは、**[Physical Model Desing]** エリアの **[Partition Type]** が**[Partitioned Table]** に設定されている場合にのみ表示されます。
- **Field Type:** すべてのフィールドに対して、**String** タイプを使用することを推奨します。
- **Date Partition Format:** パーティションフィールドが日付 (データタイプは **String)** の場合、**yyyymmmdd** といった日付形式を選択または入力します。
- **Date Partition Granularity:** たとえば、**[Day]**、**[Month]**、**[Hour]** です。 必要に応じて、パーティション細分性 を設定します。 デフォルトでは、複数の パーティション細分性 が必要な場合、細分性が高いほど、パーティションのレベルが高くなります。 たとえば、**3** つのパーティション (時、日、月) が存在する場合、複数のパーティションのリレーションシップは、**level-1** パーティション (月)、**level-2** パーティション (日)、**level-3** パーティション (時) です。

テーブルを送信します。

テーブル構造情報を編集した後、新規テーブルを開発環境と運用環境へ送信します。

- **[Load from Development Environment]** をクリックします。 テーブルが開発環境へ送信されると、ボタンはハイライトされます。 ボタンをクリックすると、開発環境で作成されたテーブルの情報は、現在のページの情報へ上書きされます。
- **[Submit to Development Environment]** をクリックします。 システムは、現在の編集ページで必要な項目がすべて設定されているかどうかを確認します。 未入力の項目があった場合、テーブルの送信を禁止するアラームが報告されます。
- **[Load from Production Environment]** をクリックします。 運用環境へ送信したテーブルの詳細情報は、現在のページの情報へ上書きされます。
- **[Create in Production Enviornment]** をクリックします。 テーブルは、運用環境のプロジェクトに作成されます。

タイプ別のテーブルの照会

[Table Management] ページでは、**[Development Environment]** または **[Production Environment]** のどちらかを選択してテーブルを照会します。 クエリ結果はフォルダーのトピック別にソートされます。

- **[Development Environment]** を選択する場合、開発環境にあるテーブルのみを照会できます。
- **[Production Environment]** を選択する場合、運用環境にあるテーブルを照会します。 運用環境にあるテーブルを操作するときは注意してください。

1.15 外部テーブル

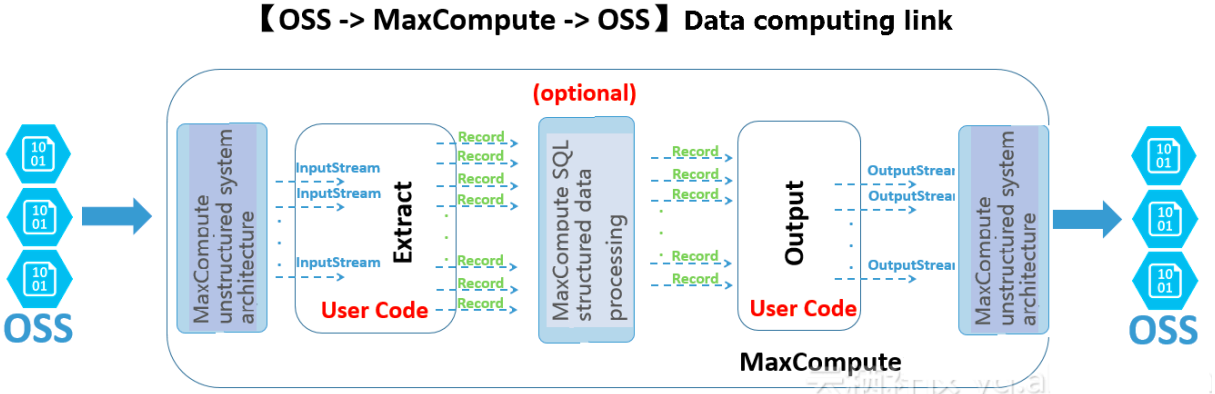
外部テーブルの概要

外部テーブルを使用する前に、次の概念について理解しておく必要があります。

名前	説明
Object Storage Service (OSS)	**OSS** は、**[Standard]**、**[Infrequent Access]**、**[Archive]** ストレージタイプをサポートします。 データストレージとアクセスに必要な要件が異なる場合のサービスシナリオに適用されます。 さらに、**OSS** は、**Apache Hadoop**、**E-MapReduce**、**BatchCompute**、**MaxCompute**、**Machine Learning Platform for AI (PAI)**、**Data Lake Anaytics**、**Function Compute**、その他 **Alibaba Cloud** サービスに対するシームレスな統合をサポートします。

MaxCompute	ビッグデータコンピューティングサービスは、高速で、データ倉庫をすべて管理するソリューションです。 **OSS** と組み合わせて使用するとき、大規模なデータを低コストで効果的に分析し処理することができます。 その処理能力から、フォレスターは、**MaxCompute** を世界有数のクラウドベースデータ倉庫の **1** つとしています。
MaxCompute の外部テーブル	この機能は、**MaxCompute v2.0** の計算フレームワークの新しい世代に基づいています。 これは、**MaxCompute** の内部テーブルへデータをロードすることなく、**OSS** に格納されているデータを直接照会します。 データを移行する際、時間と労力のみでなく、重複データのストレージコストも節約します。 **MaxCompute** の外部テーブルを使って、同様の方法で **Table Store** に格納されているデータを照会します。

次の図では、外部テーブルの処理アーキテクチャを示します。



現在、**MaxCompute** では **OSS** や **Table Store** といった非構造データのストレージでの外部テーブルの処理をサポートします。 データのフローと処理ルールに基づいて、非構造データ処理フレームワークの主な機能がデータをインポートしエクスポートし、**MaxCompute** の入出力を接続することを理解できます。 次の例では、**OSS** にある外部テーブルに適用される処理ルールを説明します。

1. **OSS** に格納されているデータは、非構造データ処理フレームワークを使って変換され、**InputStream Java** クラスを使ったユーザー定義のインターフェイスへ送られます。 抽出ルールを実装するには、入力ストリームを読み込み、解析、変換、計算する必要があります。データはレコード形式に戻す必要があります。この形式は **MaxCompute** の一般的な形式です。
2. これらのレコードは、**MaxCompute** に内蔵されている **SQL** エンジンに基づいて、新しいレコードを生成するために構造データ処理に使用されます。
3. **Output Stream Java** クラスを使ってレコードデータが出力され、**MaxCompute** によって **OSS** へインポートされる前に更に計算を実行することができます。

MaxCompute によって提供される DataWorks を使って GUI で外部テーブルを作成、検索、照会、設定、処理、および分析することができます。

ネットワークとアクセス権限の付与

MaxCompute は OSS から独立しているため、異なるクラスターでのネットワーク接続は、OSS に格納されているデータへアクセスするための MaxCompute の能力に影響する場合があります。 プライベートエンドポイント (`-internal.aliyuncs.com` で終わる) を使って、MaxCompute から OSS に格納されているデータへアクセスすることを推奨します。

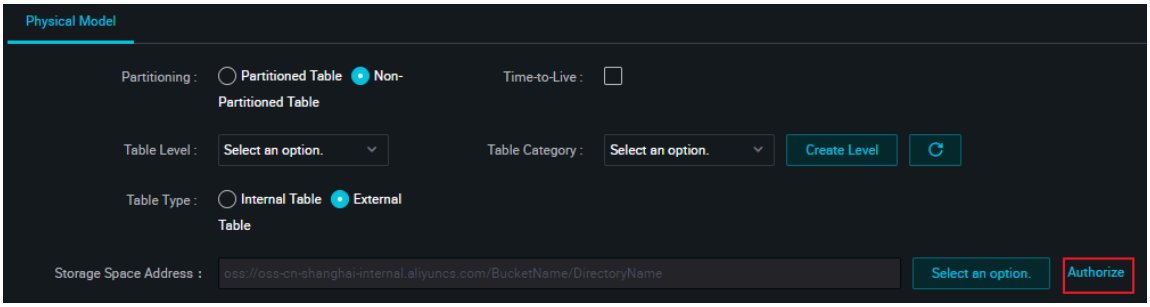
OSS に格納されているデータへアクセスするには、MaxCompute への権限が必要となります。 MaxCompute は、Alibaba Cloud が提供する Resource Access Management (RAM) と Security Token Service (STS) を使ったセキュアなデータアクセスを保証します。 テーブル作成者として、MaxCompute に対してSTS トークンを申請します。 したがって、MaxCompute と OSS は、同じ Alibaba Cloud アカウントで使用する必要があります。 同様の権限付与プロセスは、Table Store に格納されているデータへアクセスする際にも適用されます。

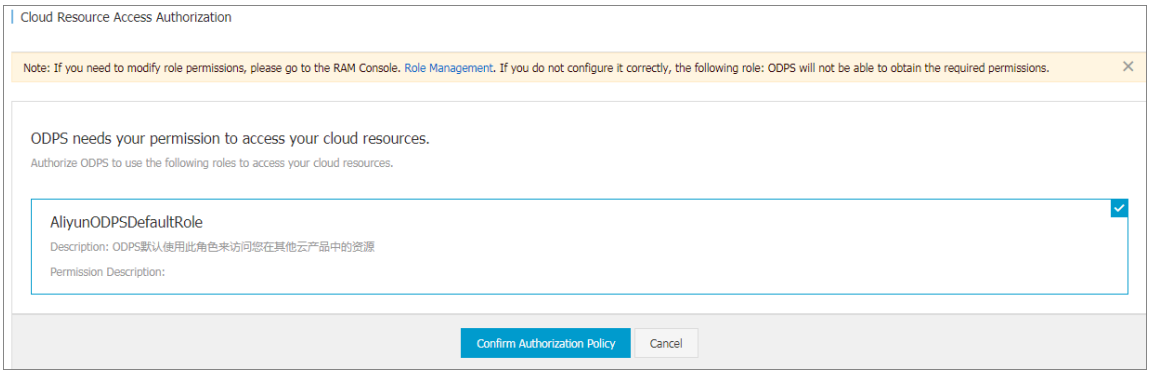
1. STS の権限付与

   MaxCompute が OSS に格納されているデータに対する直接的なアクセス権を必要とする場合、まず RAM ユーザーに対し OSS アクセス権を付与する必要があります。 Security Token Service (STS) は Alibaba Cloud が提供するセキュリティトークン管理サービスです。 Resource Access Management (RAM) に基づいたプロダクトです。 権限付与された RAM ユーザーは、カスタム有効性と共にトークンを発行し、STS からアクセスすることができます。 アプリケーションは、Alibaba Cloud API を直接呼び出し、リソースを操作することができます。 詳細については、「*OSS アクセスへの STS の権限の付与*」 をご参照ください。 次の方法のいずれかを使ってアクセス権限を付与します。

   - MaxCompute と OSS が 同じ Alibaba Cloud アカウントである場合、ログインし、ワンクリック権限付与を行います。 次の図に示すとおり、[Data Development] と [Create

Table] をクリックして、ワンクリック権限付与ページへ移動します。 詳細は、「」をご参照ください。





- カスタム権限の付与 まず、**RAM** を使って **OSS** へアクセスする権限を **MaxCompute** へ付与します。 **RAM** コンソールへログインします (**MaxCompute** と **OSS** が違う **Alibaba Cloud** アカウントである場合、そのアカウントを使って **OSS** へログインします)。 **[Role Management]** ページへ移動し、**[Create Role]** をクリックします。 **[Role Name]** の値を「**AliyunODPSDefaultRole**」または「**AliyunODPSRoleForOtherUser**」に設定します。

  [Role Details] を設定します。

```
--When MaxCompute and OSS are under the same Alibaba Cloud account
.
{
"Statement": [
{
"Action": "sts:AssumeRole",
"Effect": "Allow",
"Principal": {
"Service": [
"odps.aliyuncs.com"
      ]
    }
  }
],
"Version": "1"
}
--When MaxCompute and OSS are under different Alibaba Cloud
accounts.
{
"Statement": [
{
"Action": "sts:AssumeRole",
```

```
"Effect": "Allow",
"Principal": {
"Service": [
"Alibaba Cloud account for MaxCompute@odps.aliyuncs.com"
      ]
    }
  }
],
"Version": "1"
}
```

[Role Authorization Policies] を設定します。**OSS** アクセス権を付与するために必要な「**AliyunODPSRolePolicy**」ポリシーを検索します。「**AliyunODPSRolePolicy**」ポリシーをロールに添付します。**[Search and Attach]** を使ってこのポリシーを探すことができない場合、**[Input and Attach]** を使ってロールに権限を付与します。「**AliyunODPSRolePolicy**」ポリシーのポリシーコンテントは次のとおりです。

```
{
  "Version": "1",
  "Statement": [
    {
      "Action": [
        "oss:ListBuckets",
        "oss:GetObject",
        "oss:ListObjects",
        "oss:PutObject",
        "oss:DeleteObject",
        "oss:AbortMultipartUpload",
        "oss:ListParts"
        ],
        "Resource": "*",
        "Effect": "Allow"
  },
  {
      "Action": [
        "ots:ListTable",
        "ots:DescribeTable",
        "ots:GetRow",
        "ots:PutRow",
        "ots:UpdateRow",
        "ots:DeleteRow",
        "ots:GetRange",
        "ots:BatchGetRow",
        "ots:BatchWriteRow",
        "ots:ComputeSplitPointsBySize"
      ],
      "Resource": "*",
      "Effect": "Allow"
    }
  ]
}
```

2. **Data Integration** での **OSS** データソースの使用

   Data Integration ですでに作成されている **OSS** データソースを直接使用することができます。

外部テーブルの作成

1. DDL 文を使ってテーブルを作成します。

   [Data Development] ページへ移動します。「テーブルの管理」を参照し、DDL 文を使ってテーブルを作成します。 ODPS 構文に従う必要があります (「テーブルの操作」 をご参照ください)。 STS 権限が付与されている場合、odps.properties.rolearn 属性を含める必要はありません。 次の例では、DDL 文を使ってテーブルを作成する方法を説明します。 文の EXTERNAL キーワードは、このテーブルが外部テーブルであることを示しています。

```
CREATE EXTERNAL TABLE IF NOT EXISTS ambulance_data_csv_external(
vehicleId int,
recordId int,
patientId int,
calls int,
locationLatitute double,
locationLongtitue double,
recordTime string,
direction string
)
STORED BY 'com.aliyun.odps.udf.example.text.TextStorageHandler' --
The STORED BY clause specifies the StorageHandler for the correspond
ing file format. This clause is required.
with SERDEPROPERTIES (
'delimiter'='\\|', --The SERDEPROPERITES clause specifies the
parameters used when serializing or deserializing data. These
parameters are passed into the code of Extractor through DataAttrib
utes. This clause is optional.
'odps.properties.rolearn'='acs:ram::xxxxxxxxxxxxx:role/aliyunodps
defaultrole'
)
LOCATION 'oss://oss-cn-shanghai-internal.aliyuncs.com/oss-odps-test/
Demo/SampleData/CustomTxt/AmbulanceData/'              --The LOCATION
 clause specifies the location of the external tables. This clause
is optional.
USING 'odps-udf-example.jar'; --The USING clause specifies the Jar
files that store the user-defined classes. This clause is optional,
depending on whether you use user-defined classes.
```

   csv または tsv ファイルに対し内蔵されているストレージハンドラーに対応するSTORED BY に続くパラメーターは次のとおりです。

   - com.aliyun.odps.CsvStorageHandler パラメーターは CSV 形式です。 CSV 形式でのデータの読み込みや書き込み方法を定義します。 形式にはコンマ (,) で区切られてい

る列と行改行文字 (\n) で終わる行が含まれます。 たとえば、 `STORED BY'com.aliyun.odps.CsvStorageHandler'` はサンプルパラメーターです。

- `com.aliyun.odps.TsvStorageHandler` パラメーターは、**TSV** 形式です。 **TSV** 形式でのデータの読み込みと書き込み方法を定義します。 形式にはタブ文字 (\t) で区切られている列と行改行文字 (\n) で終わる行が含まれます。

`STORED BY` に続くパラメーターは、**TextFile**、 **SequenceFile**、**RCFile**、 **AVRO**、 **ORC**、および **Parquet** といった **open-source file formats** に対するストレージハンドラーの指定もサポートします。 **TextFile** 形式では、**SerDe** クラスを指定します。 たとえば、`org.apache.hive.hcatalog.data.JsonSerDe` です。

- **org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe -> stored as textfile**
- **org.apache.hadoop.hive.ql.io.orc.OrcSerde -> stored as orc**
- **org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe -> stored as parquet**
- **org.apache.hadoop.hive.serde2.avro.AvroSerDe -> stored as avro**
- **org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe -> stored as sequencefile**

For external tables that are in the open-source formats, the statements to create tables are as follows.

```
CREATE EXTERNAL TABLE [IF NOT EXISTS] (<column schemas>)
[PARTITIONED BY (partition column schemas)]
[ROW FORMAT SERDE '']
STORED AS
[WITH SERDEPROPERTIES ( 'odps.properties.rolearn'='${roleran}'
[,'name2'='value2',...]
) ]
LOCATION 'oss://${endpoint}/${bucket}/${userfilePath}/';
```

SERDEPROPERTIES 句の属性は次の表に示すとおりです。 現在、**OSS** で **CSV** および **TST** ファイルから **gzip** 形式で圧縮されたデータに対して、**MaxCompute** は内蔵のエクストラクターを使った読み取りのみをサポートしています。 ファイルを **gzip** 圧縮にするかどうかを選択します。 属性設定はファイル形式によって異なります。

属性	値	デフォルト値	説明
odps.text.option.gzip.input.enabled	**true/false**	**false**	圧縮データの読み取りを有効、無効にします。

odps.text.option.gzip.output.enabled	**true/false**	**false**	圧縮データの書き込みを有効、無効にします。
odps.text.option.header.lines.count	**N (非負整数)**	**0**	ファイルの最初の **N** 行をスキップします。
odps.text.option.null.indicator	**String**	**""**	**NULL** を文字列の値に置換します。
odps.text.option.ignore.empty.lines	**true/false**	**true**	空白行を無視するかどうかを指定します。
odps.text.option.encoding	**UTF-8/UTF-16/US-ASCII**	**UTF-8**	ファイルのエンコード設定を指定します。

LOCATION 句は、外部テーブルの格納アドレスを指定します。形式は、**oss://oss-cn-shanghai-internal.aliyuncs.com/BucketName/DirectoryName** です。 ダイアログボックスを使って **OSS** にディレクトリを選択します。 ファイルは選択しないでください。

[Tables] タブのノードディレクトリで **DDL** 文を使って作成したテーブルを検索します。**Level 1 Topic** または **Level 2 Topic** を変更して、テーブルのディレクトリを変更します。

2. **Table Store** の外部テーブル

   Table Store で外部テーブルを作成する文は次のとおりです。

```
CREATE EXTERNAL TABLE IF NOT EXISTS ots_table_external(
odps_orderkey bigint,
odps_orderdate string,
odps_custkey bigint,
odps_orderstatus string,
odps_totalprice double
)
STORED BY 'com.aliyun.odps.TableStoreStorageHandler'
WITH SERDEPROPERTIES (
'tablestore.columns.mapping'=':o_orderkey,:o_orderdate,o_custkey,
o_orderstatus,o_totalprice', -- (3)
'tablestore.table.name'='ots_tpch_orders'
'odps.properties.rolearn'='acs:ram::xxxxx:role/aliyunodpsdefaultrole
'
)
```

```
LOCATION 'tablestore://odps-ots-dev.cn-shanghai.ots-internal.
aliyuncs.com';
```

説明:

- com.aliyun.odps.TableStoreStorageHandler は **MaxCompute** ビルトインストレージハンドラーで、**Table Store** のデータを処理します。
- SERDEPROPERITES は、パラメーターのオプションを提供します。

  TableStoreStorageHandler を使って「**tablestore.columns.mapping**」と「**tablestore.table.name**」を指定する必要があります。

  - tablestore.columns.mapping**:** このパラメーターは必要です。 **MaxCompute** がアクセスする **Table Store** のテーブルの列を説明します。主キー列とプロパティ列を含みます。 主キー列は、列名の前にコロン **(:)** で示します。 この例では、主キー列は、**p:o_orderkey** と **:o_orderdate** です。 他は、プロパティ列です。 **Table Store** は最大 **4** つの主キー列をサポートします。 データタイプには、**String**、**Integer**、**Binary** です。 主キーの **1** 番目の列は、パーティションキーです。 マッピングを指定する際、**Table Store** にあるテーブルのすべての主キー列を指定します。 **MaxCompute** がすべてのプロパティ列を指定する代わりにアクセスするプロパティ列を指定する必要があります。
  - tablestore.table.name**: Table Store** にアクセスするテーブル名 テーブル名が **Table Store** にない場合、エラーが報告されます。 **MaxCompute** は **Table Store** にテーブルを作成しません。
- LOCATION**: Table Store** インスタンスの名前とエンドポイントを指定します。

3. **GUI** でテーブルを作成します。

   [Data Development] ページへ移動し、「テーブルの管理」を参照し **GUI** でテーブルを作成します。 外部テーブルには次の属性が含まれます。

   ・ 基本的な属性

   - **Table name** (テーブルを作成し名前を入力します)
   - **Table alias**
   - **Level 1 Topic** と **Level 2 Topic**
   - **Description**

   ・ **Physical model**

   - **Table type:** 外部テーブルを選択します。
   - **Partition: Table Store** にある外部テーブルはパーティション化をサポートしません。
   - **[Select the memory address]:** `LOCATION` 句を指定します。 次の図に示すとおり、`LOCATION` 句を **[Physical model]** セクションで指定します。 ダイアログボックスで外部テーブルの格納場所をクリックし選択します。 **[One-click authorization]** を実行します。
   - **[Select storage format]:** 必要とされるファイル形式を選択します。
     CSV、**TSV**、**TextFile**、**SequenceFile**、**RCFile**、**AVRO**、**ORC**、および **Parquet** とカスタムファイル形式がサポートされています。 カスタムファイル形式を選択する場

合、対応するリソースを選択する必要があります。 リソースを送信すると、リソースからクラスが自動的に解析されます。 クラス名を選択します。

- `rolearn`: STS 権限が付与されている場合、この rolearn 属性を指定する必要はありません。

・ **Table structure design**

- **Data type:** MaxCompute 2.0 ではフィールドに対して、INYINT、SMALLINT、INT、BIGINT、VARCHAR および STRING タイプをサポートしています。
- **Actions:** フィールドを作成、変更、削除します。
- **Length/Set:** VARCHAR タイプの列の最大長を指定します。 コンポジットデータタイプでは、定義を入力することができます。

サポートしているデータタイプ

外部テーブルでサポートされている基本データタイプは次の表に示すとおりです。

データ型	新規	例	説明
TINYINT	可	**1Y**，**-127**	署名された **8** ビットの整数で、範囲は **-128** から **127**です。
SMALLINT	可	**32767S**，　**-100S**	署名された **16** ビットの整数で、範囲は **-32,768** から **32,767** です。
INT	可	**1000, -15645787**	署名された **32** ビットの整数で、範囲は **-231** から **231-1** です。
BIGINT	不可	**100000000000L, -1L**	署名された **64** ビットの整数で、範囲は **-263 + 1** から **263 - 1** です。
FLOAT	可	なし	**32** ビットのバイナリ浮動小数点です。

DOUBLE	不可	**3.1415926 1E+7**	**8** ビットの倍精度浮動小数点です **(64** ビットバイナリ浮動小数点**)**
DECIMAL	不可	**3.5BD, 99999999999. 9999999BD**	**10** 進の抽出数値です。 精度範囲は**-1036 + 1** から **1036 -1**で、スケールは **10** から **18** です。
VARCHAR(n)	可	なし	変数長の文字列です。 長さは **n** です。範囲は、**1** から **65535** です。
STRING	不可	**“abc”、’ bcd ’、” alibaba”**	文字列 現在、最大長は **8M** です。
BINARY	可	なし	バイナリ数値です。 現在、最大長は **8M** です。
DATETIME	不可	**DATETIME ‘2017-11-11 00:00:00’**	日時と時間のデータタイプです。 **UTC-8** はシステムの標準時間として使用されています。 範囲は **0000-01- 01** から **9999-12-31** で、ミリ秒で正確です。
TIMESTAMP	可	**TIMESTAMP ‘2017-11-11 00:00:00. 123456789’**	**TIME STAMP** データタイプで、タイムゾーンと独立したものです。 範囲は **0000-01- 01** から **9999-12-31** で、ナノ秒で正確です。
BOOLEAN	不可	**TRUE、FALSE**	論理ブール **(TRUE/FALSE)**

外部テーブルでサポートされているコンポジットデータタイプは次の表に示すとおりです。

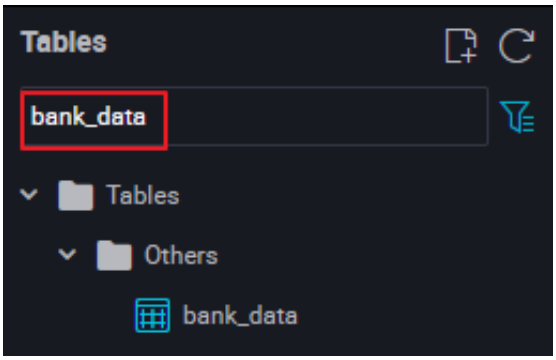
タイプ	説明	コンストラクター
ARRAY	**array< int >; array< struct < a:int, b:string >>**	**array(1, 2, 3); array(array(1, 2); array(3, 4))**
MAP	**map< string, string >; map < smallint, array< string>>**	**map(“k1” , “v1” , “k2 ” , “v2”); map(1S, array (‘a’ , ‘b’), 2S, array(‘ x’ , ‘y))**
STRUCT	**struct< x:int, y:int>; struct < field1:bigint, field2: array< int>, field3:map< int, int>>**	**named_struct(‘x’ , 1, ‘ y’ , 2); named_struct(‘ field1’ , 100L, ‘field2’ , array(1, 2), ‘field3’ , map (1, 100, 2, 200))**

MaxCompute 2.0 で新しくサポートされたデータタイプ **(TINYINT、SMALLINT、 INT、FLOAT、VARCHAR、TIMESTAMP 、BINARY** またはコンポジットデータタイプ**)** を使用す

る場合、テーブルを作成する文の前に `set odps.sql.type.system.odps2=true;` を含めます。 文を送信、実行し、設定文を使ってテーブルを作成します。 **HIVE** との互換性が必要な場合、`odps.sql.hive.compatible=true;` 文を含めます。

外部テーブルの表示と処理

Tables ビューで外部テーブルを探します。

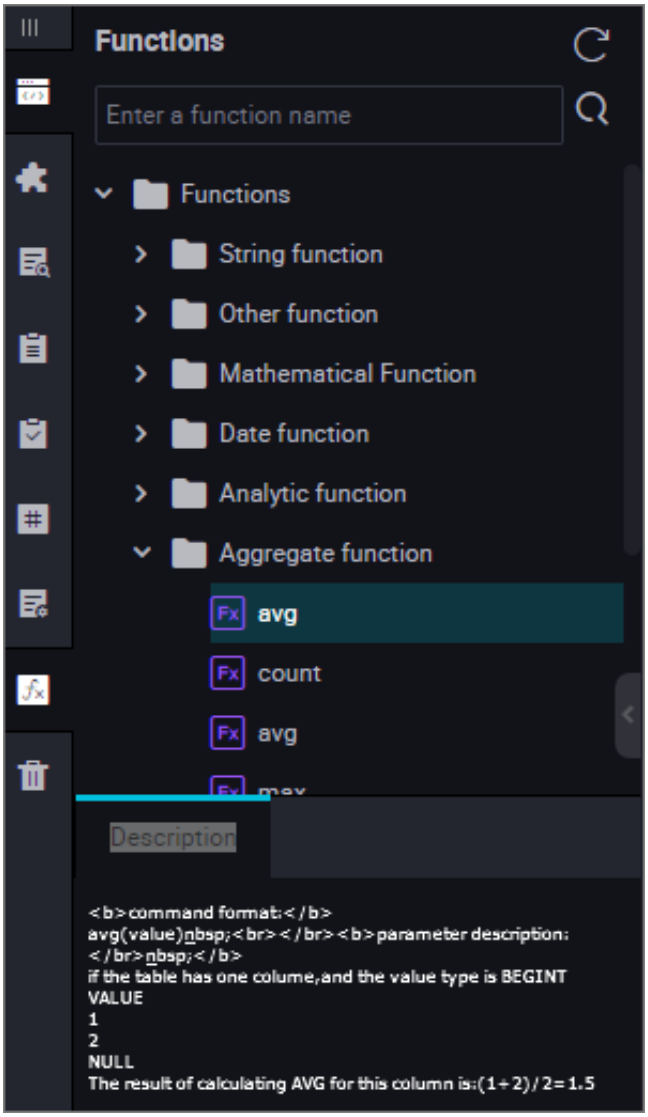


外部テーブルの処理は、内部テーブルの処理と似ています。 外部テーブルに関する詳細は、「テーブルの管理」と「*#unique_64*」をご参照ください。

1.16 関数

関数の一覧では、現在利用可能な関数、関数の分類、関数の使用説明およびインスタンスを提供します。

関数の一覧には **6** つの部分があり、その他の関数、文字列処理関数、数学関数、日付関数、ウィンドウ関数、および集約関数が含まれています。 これらの関数はシステムによって提供されています。 関数をドラッグして、その説明と使用方法を表示します。



1.17 エディターショートカット一覧

コード編集用の共通ショートカット

Windows クロームバージョン

`Ctrl + S` 保存

`Ctrl + Z` 元に戻す

`Ctrl + Y` やり直し

`Ctrl + D` 同じワードを選択

`Ctrl + X` 行の切り取り

`Ctrl+Shift+K` 行の削除

`Ctrl + C` 現在の行をコピー

Ctrl+i 行を選択

Shift+Alt+マウスをドラッグ 列モード編集、この部分のすべてのコンテンツを編集

Alt + マウス 複数列モード編集、マルチラインインデント

Ctrl + Shift + L 同一のすべての文字列インスタンスにカーソルを追加、バッチ変更

Ctrl + F 検索

Ctrl + H 置換

Ctrl + G 特定の行を検索

Alt + Enter 検索で一致するすべてのキーワードを選択

Alt↓ / Alt↑ 現在の行を上下に移動

Shift + Alt + ↓ / Shift + Alt + ↑ 現在の行を上下にコピー

Shift + Ctrl + K 現在の行を削除

Ctrl + Enter / Shift + Ctrl + Enter カーソルを上下に移動

Shift + Ctrl + \ 一致するブラケットへカーソルを移動

Ctrl +] / Ctrl + [インデントの追加/削除

Home / End 現在の行の先頭または終わりへ移動

Ctrl + Home / Ctrl + End 現在のファイルの先頭/終わりへ移動

Ctrl + → /Ctrl + ← 単語ごとにカーソルを左右へ移動

Shift + Ctrl + [/ Shift + Ctrl +] カーソルでポイントしたブロックを表示/非表示

Ctrl + K + Ctrl + [/ Ctrl + K + Ctrl +] カーソルでポイントしたサブブロックを表示/非表示

Ctrl + K + Ctrl + 0 / Ctrl + K + Ctrl + j すべてのエリアを折りたたむ/展開する

Ctrl + / カーソルがある行またはコードブロックにコメントを書き込む/消去する

MAC **クロームバージョン**

cmd + S 保存

cmd + Z 元にもどす

cmd + Y やり直し

`cmd+D` 同じワードを選択

`cmd + X` 行の切り取り

`cmd + shift + K` 行の削除

`cmd + C` 現在の行をコピー

`cmd +i` 現在の行を選択

`cmd + F` 検索

`cmd + alt + F` 置換

`alt↓ / alt↑` 現在の行を上下に移動

`shift + alt + ↓ / shift + alt + ↑` 現在の行を上下にコピー

`shift + cmd + K` 現在の行を削除

`cmd + Enter / shift + cmd + Enter` カーソルを上下に移動

`shift + cmd + \` 一致するブラケットへカーソルを移動

`cmd + ] / cmd + [` インデントの追加/削除

`cmd + ← / cmd + →` 現在の行の先頭/終わりへ移動

`cmd + ↑ / cmd + ↓` 現在のファイルの先頭/終わりへ移動

`alt + → /alt + ←` 単語ごとにカーソルを左右に移動

`alt + cmd + [ / alt + cmd + ]` カーソルでポイントしているブロックを表示/非表示

`cmd + K + cmd + [ / cmd + K + cmd + ]` カーソルでポイントしているサブブロックを表示/非表示

`cmd + K + cmd + 0 / cmd + K + cmd + j` すべてのエリアを折りたたむ/展開する

`cmd + /` カーソルがある行またはコードブロックにコメントを書き込む/消去する

複数カーソル/選択

`alt + Clicking with the mouse` カーソルの挿入

`alt + cmd + ↑/↓` カーソルを上下に挿入

`cmd + U` 最後のカーソル操作を元に戻す

`shift + alt + I` カーソルを選択したコードブロックの各行の最後に挿入

`cmd + G／shift + cmd + G` 次/前の項目を検索

`cmd + F2` マウスが選択したすべての文字を選択

`shift + cmd + L` マウスが選択したすべての部分を選択

`alt+Enter` 検索で一致したキーワードをすべて選択

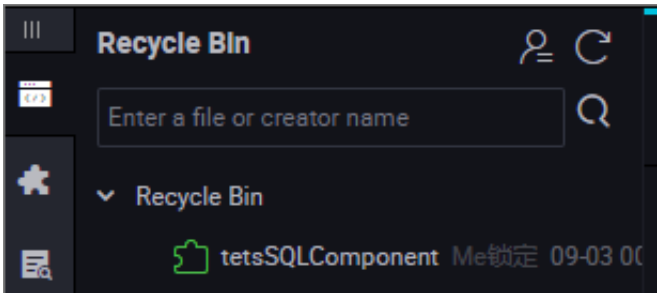
`shift + alt + マウスをドラッグ` 編集する複数の列を選択

`shift + alt + cmd + ↑ / ↓` 編集する複数の列を選択するため、カーソルを上下に移動

`shift + alt + cmd + ← / →` 編集する複数の列を選択するため、カーソルを左右に移動

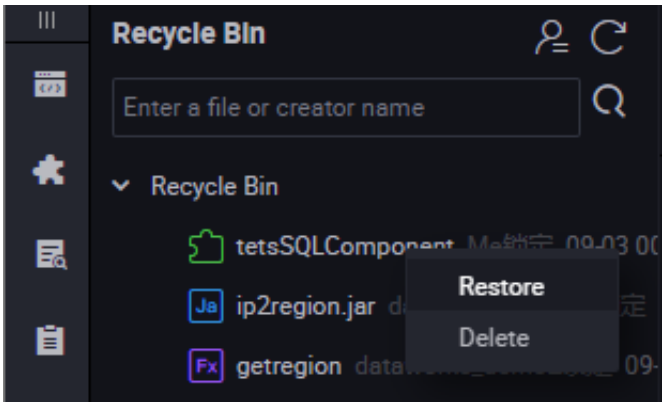
1.18 ゴミ箱

DataWorks には独自のゴミ箱があります。ページの左上隅にある **[Recycle Bin]** をクリックします。



[Recycle Bin] ページでは、現在のプロジェクトで削除されたすべてのノードを確認することができます。 ノードを右クリックして、元に戻す、または完全に削除することができます。

[Recycle Bin] ページの右側にある **[Show My Files]** をクリックして、削除されたノードを表示します。



注：

ノードがゴミ箱から完全に削除された場合、元に戻すことはできません。

2 DataService Studio

2.1 DataService Studio の概要

DataService Studio は、企業がプライベートおよびパブリックの **API** を集中管理できるよう、データサービスのバスを構築することを目的としています。 **DataService Studio** を使用することで、データテーブルに基づいた **API** をすばやく作成することができます。また、一元的な管理とリリースを目的として、**DataService Studio** プラットフォームに既存の **API** を登録することができます。 なお、**DataService Studio** は **API Gateway** に接続されています。 ワンクリックで **API Gateway** に **API** をデプロイすることができます。 **DataService Studio** は、**API Gateway** と連携することで、安全かつ安定した、低コストで使いやすいデータの共有サービスを提供します。

DataService Studio はサーバーレスアーキテクチャーを採用しています。 ユーザーは、実行環境などのインフラストラクチャなどに気を配る必要はなく、**API** のクエリロジックを気にする必要があるだけです。 **DataService Studio** では、ユーザー用のコンピューティングリソースを準備しています。また、弾力的なスケーリングをサポートしており、**O&M** コストがかかりません。

データ API の作成

現在、**DataService Studio** では、リレーショナルデータベースおよび **NoSQL** データベースのテーブルに基づいたデータ **API** をすばやく作成するため、視覚化されたウィザードの使用をサポートしています。 コードを記述することなく、数分でデータ **API** を設定することができます。 高度なユーザーの、パーソナライズされたクエリ要件を満たすため、**DataService Studio** では、カスタマイズ **SQL** スクリプトモードを提供しています。この機能により、 **API** クエリの **SQL** 文をユーザー自身でコンパイルすることができます。 また、複数テーブルの関連付け、複雑なクエリ条件、集計関数もサポートされています。

API の登録

DataService Studio では、**DataService Studio** で登録した既存の **API** サービス、およびデータテーブルに基づいて作成された **API** の集中管理もサポートしています。 現在登録できるのは **RESTful API** のみになります。 サポートされているリクエストメソッドには、**GET**、**POST**、**PUT**、および **DELETE** があります。 サポートされているデータ型は、**JSON** 形式データと **XML** 形式データです。

API Gateway

API Gateway では、**API** のパブリッシュ、管理、メンテナンス、および **API** サブスクリプション期間の管理などの **API** 管理サービスを提供します。 **API Gateway** では、パートナーや開発者を対象として、マイクロサービスの統合、フロントエンドとバックエンドの分離、およびシステム統合を実装するための、シンプルかつ高速で低コスト低リスクのメソッドを提供します。

DataService Studio は **API Gateway** に接続されています。 管理を行うため、ユーザーは、**API** の権限付与と認証、トラフィック制御、メータリングなど、**DataService Studio** で作成および登録されたすべての **API** を **API Gateway** にデプロイすることができます。

API Market

Ali cloud の **API market** は、金融、人工知能、電子商取引、交通地理学、生活サービス、企業管理、および **8** つのパブリックアフェアーズ主要カテゴリーを網羅する、中国で最も総合的な **API** 売買市場です。ここでは、数千の **API** 製品がオンラインで販売されています。

ユーザーの各種 **API** を、**DataService Studio** から **API Gateway** にパブリッシュした後、それらを **Alibaba Cloud API Marketplace** に公開することができます。 この方法により、ユーザーの会社は経済的利益を簡単に得ることができます。

2.2 用語集

データサービスに関連する単語について、以下に説明します。

- データソース**:** データベースのリンクです。 データサービスでは、データソースを経由してデータにアクセスします。 データソースは **Data Integration** 内でのみ、設定を行えます。
- **API** の作成**:** データテーブルに基づいて **API** を作成します。
- **API** の登録**:** 統合管理用に既存の **API** を **Data Service** に登録します。
- ウィザード**: API** 作成の手順を案内します。 このメソッドはシンプルな **API** を作成したい初心者に適しています。 ユーザーはコードを記述する必要はありません。
- スクリプト**: SQL** スクリプトを記述し、**API** を作成します。 このメソッドは、テーブル結合クエリ、複雑なクエリ、集計関数をサポートしています。 このメソッドは、複雑な **API** を作成したい経験豊富な開発者に適しています。
- **API** グループ**: API** グループは、特定のシナリオまたは特定のサービスを利用するための **API** セットです。 **API** グループは、データサービス内での最小のグループ単位、および **API Gateway** によって管理される最小の単位です。 **API** グループは、**API** 製品として **Alibaba Cloud API Market** に公開されています。

- **API Gateway: API** を管理するため、**Alibaba Cloud** によって提供されるサービスです。**API Gateway** は、**API** のサブスクリプション期間管理、権限管理、アクセス管理、およびトラフィック制御をサポートしています。
- **API Market: Alibaba Cloud API Market** は、**Alibaba Cloud Market** 上に設置された、国内で最も完璧かつ統合的な **API** 取引のプラットフォームです。

2.3 API の生成

2.3.1 データソースの設定

データ **API** を使用してサービスを生成する前に、データソースを事前に設定する必要があります。 データサービスを使用することで、データソースからデータテーブルのスキーマ情報を取得し、**API** リクエストを処理することができます。

Dataworks コンソールの **[data integration]** > **[data source]** ページにて、データソースを設定します。各種データソースタイプのサポートおよびそれらの設定方法を次の表に示します。

データソース名	データ API 生成のウィザードモード	データ API 生成のスクリプトモード	設定方法
RDS (ApsaraDB for RDS)	対応	対応	**RDS** には、**MySQL**、**PostgreSQL**、および **SQL Server** が含まれています。
DRDS	対応	対応	*#unique_73*
MySQL	対応	対応	*#unique_74*
PostgreSQL	対応	対応	*#unique_75*
SQL Server	対応	対応	*#unique_76*
Oracle	対応	対応	*#unique_77*
AnalyticDB (ADS)	対応	対応	*#unique_78*
Table Store (OTS)	対応	非対応	*#unique_79*
MongoDB	対応	非対応	*#unique_80*

2.3.2 API 生成の概要

現在、データサービスでは、視覚的に設定されたウィザードモードにより、リレーショナルデータベースおよび **NoSQL** データベースによる、テーブルの高速生成をサポートしています。データ **API** では、データ **API** を設定するため、ものの数分でプログラムを書くような能力は必要あ

りません。 高度なユーザーの、パーソナライズされたクエリ要件を満たすため、データサービスにはカスタマイズ SQL スクリプトモードが用意されています。これにより、API クエリの SQL 文を自身でコンパイルですることができます。 また、複数テーブルの関連付け、複雑なクエリ条件、集計関数もサポートされています。

ウィザードモードおよびスクリプトモードの機能は、以下のとおりです。

機能	特徴	ウィザードモード	スクリプトモード
オブジェクトの照会	1 つのデータソースから、1 つのデータテーブルを照会します。	対応	対応
	1 つのデータソースから、複数の結合テーブルの照会	非対応	対応
フィルターバー	正確な数値を照会します。	対応	対応
	数値範囲の照会	非対応	対応
	正確な文字列との突き合わせ	対応	対応
	文字列のあいまい検索	対応	対応
	必須 / オプションのパラメーターの設定	対応	対応
クエリの結果	フィールドの値を返します。	対応	対応
	フィールド値の数学的計算結果を返す	非対応	対応
	フィールド値の集計計算結果を返す	非対応	対応
	ページネーションによる結果の表示	対応	対応

2.3.3 ウィザードモードでの API の生成

本ページでは、ウィザードモードでの API 生成の手順および考慮事項について説明します。

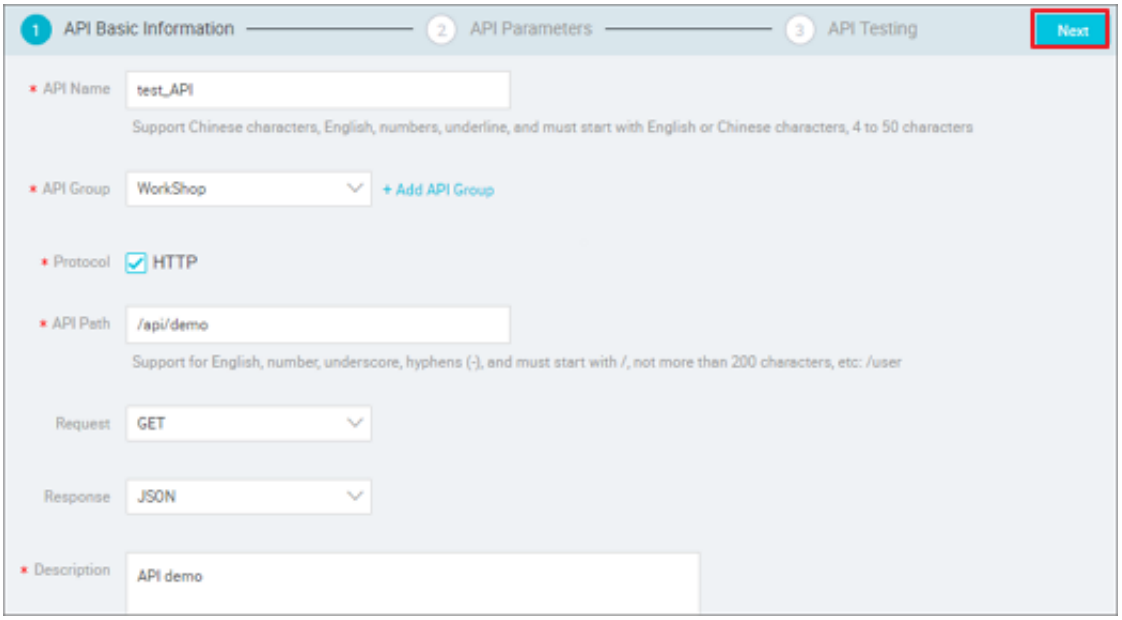
ウィザードモードを使用してデータを生成すると、コードを記述することなく、簡単に API を使用することができます。プロダクトインターフェイスから設定を確認することにより、API をすばやく生成することができます。 API の機能に対する高度な要件を持たないユーザー、またはコード開発経験の少ないユーザーは、このウィザードを使用することを推奨します。

 注：

API の設定を行う前に、Dataworks コンソールの[Data integration] > [Data Source] ページにて、データソースを設定します。

API 基本情報の設定

1. **[API Service list]** > **[Generate API]** に移動します。
2. **[Wizard Mode]** をクリックし、**API** の基本事項を入力します。



設定では、 **API** グループ化の設定に注意してください。 **API** グループには、特定のシナリオに使用される各種 **API** 集が含まれています。 これは **API Gateway** での最小管理単位です。 **Alibaba Cloud API** マーケットでは、各 **API** グループは特定の **API** に対応しています。

> 注：
> **API** グループ化の設定例は次のとおりです。
>
> たとえば、天気を照会するのに **API** プロダクトの設定を行いたい場合、都市名検索による天気検索 **API**、観光地天気検索 **API**、郵便番号検索による天気 **API** の **3** 種類の **API** プロダクトを設定し、**"**天気の照会**"** と名付けた **API** グループを作成し、このグループに上記の **3** つの **API** を配置します。 その **API** がマーケットに公開されると、天気の照会プロダクトとして表示されます。
>
> もちろん、ユーザーが生成した **API** がユーザー自身のアプリ内で使用される場合、分類するのにグループ化を行うことができます。

現在、ビルド **API** は **HTTP** プロトコル、**GET** リクエストモード、および **JSON** リターンのみをサポートしています。

3. **API** の基本情報を入力したら、**[Next]** をクリックし、**API** パラメーター設定ページに移動します。

API パラメーターの設定

1. **[Data source type]** > **[Data source name]** > **[Table]** に移動し、設定するテーブルを選択します。

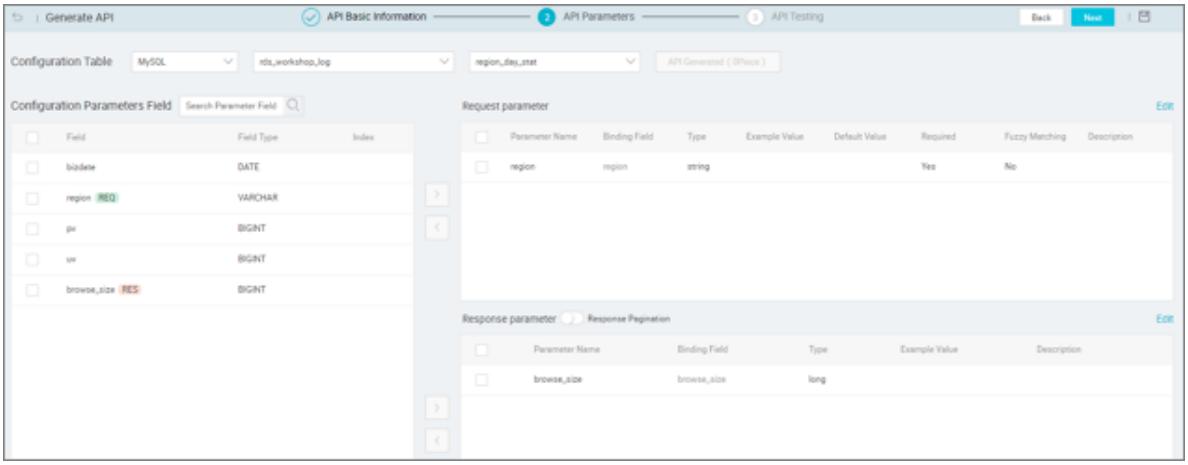
   > 注：
   > データセット内でデータソースを事前に設定する必要があります。データテーブルのドロップダウンボックスは、テーブル名検索に対応しています。

2. 次に、リクエストとレスポンスのパラメーターを指定します。

   データテーブルが選択されると、テーブルのすべてのフィールドが左側に表示されます。 リクエストパラメーターとレスポンスパラメーターとして使用するフィールドを選択し、それらを対応するパラメーターリストに追加します。

3. 最後に、パラメーター情報を編集して完了です。

   リクエストおよびリターンパラメータリスト右上の、**[Edit]**をクリックし、パラメーター情報の編集ページに移動します。パラメーターの名前、サンプル値、既定、必須、あいまい一致 (文字列タイプのみサポート) の設定を行います。 オプションと説明のフィールドは必須です。



設定プロセスでは、ページング結果の戻り設定に注意を払う必要があります。

- **[Response pagination]** を有効にしない場合、既定では、**API** は最大 **500** レコードを出力します。
- 返される結果が **500** を超える可能性がある場合、**[Response pagination]** 機能をオンにします。

次のパブリックパラメーターは、**[Response pagination]** 機能が有効になっている場合にのみ、使用できます。

- 共通のリクエストパラメーター
  - **pageNum:** 現在のページ番号です。
  - **Pagesize:** ページサイズです。 つまり、**1** ページごとのレコード数です。
- 共通のレスポンスパラメーター
  - **pageNum:** 現在のページ番号です。
  - **Pagesize:** ページサイズです。 つまり、**1** ページごとのレコード数です。
  - **totalNum:** レコードの合計数です。

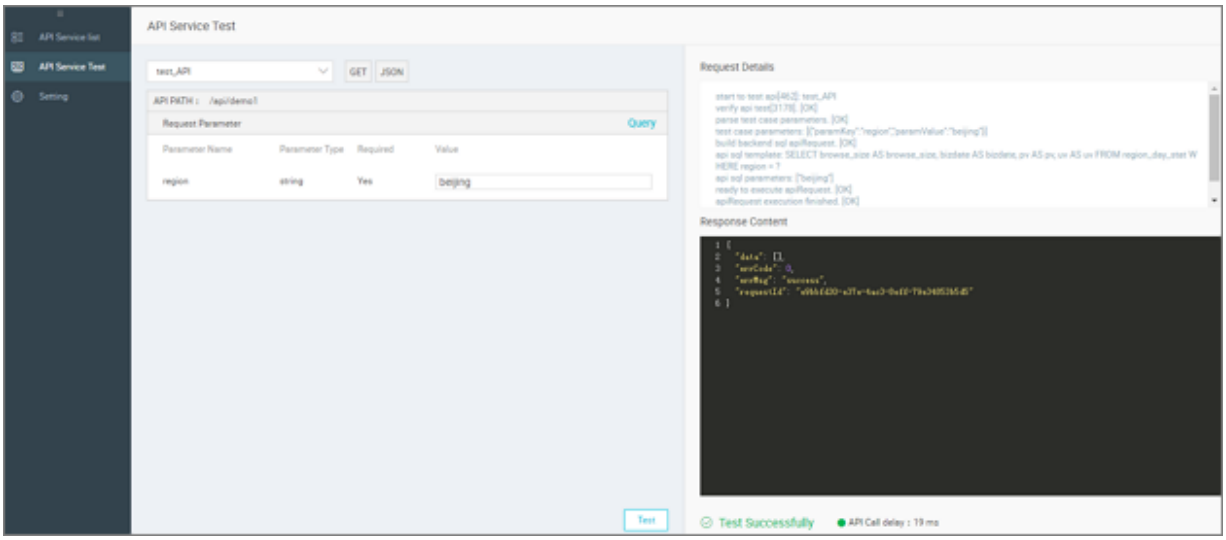
注 :

- リクエストパラメーターは、同等のクエリのみをサポートし、リターンパラメーターは、現状の値のフィールド出力のみをサポートします。
- 可能な限り、インデックスフィールドをリクエストパラメーターに設定します。
- **API** にはリクエストパラメーターを指定することはできません。 その場合、ページネーション機能を有効にする必要があります。
- **API** の呼び出し元にとって、**API** の詳細を理解しやすくするため、**API** のサンプル値、既定値、および説明パラメーターを指定することを推奨します。
- 現在のテーブルで生成されている **API** のリストを表示するには、構成済みの **API** をクリックします。同じ **API** が生成されないようにします。

API パラメーターの設定が完了したら、**[Next]** をクリックし、**API** のテストセクションに移動します。

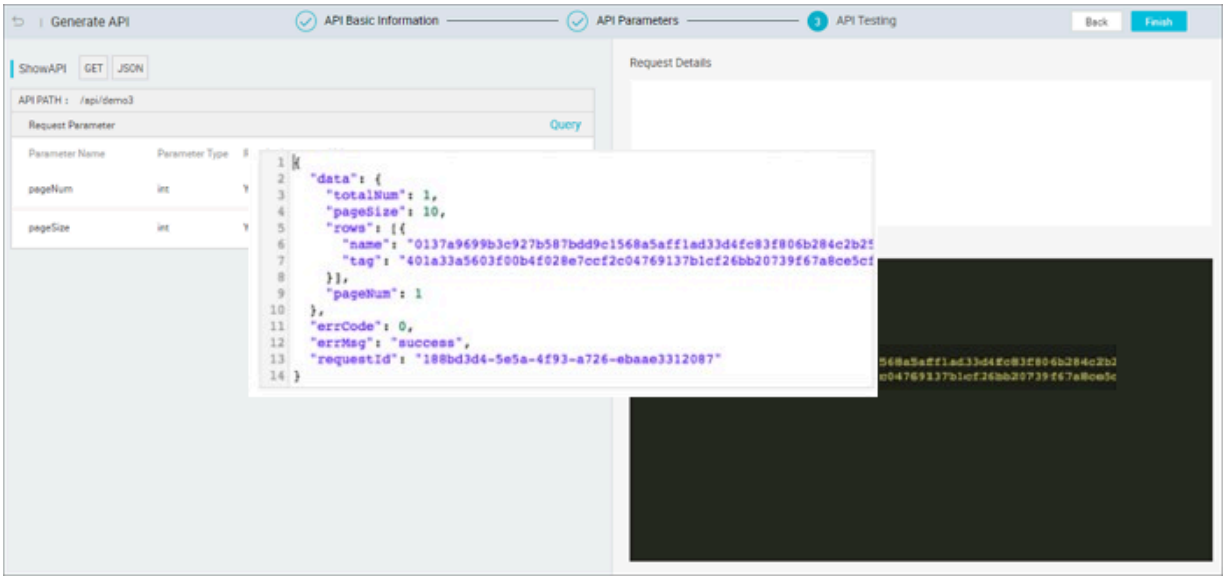
API のテスト

API パラメーターの設定が完了したら、**API** のテストを開始します。



パラメーターを設定し、**[Start Test]** をクリックして、**API** リクエストをオンラインで送信します。 **API** リクエストの詳細と応答結果が右側に表示されます。 テストが失敗した場合は、エラーメッセージを注意深く読み、適切な調整を行って **API** を再テストします。

設定プロセスでは、標準の応答例の設定に注意する必要があります。 **API** をテストするとき、システムは自動的にエラー例およびエラーコードを生成します。 ただし、標準の応答例は自動的には生成されません。 テストが成功したら、**[Save as Normal Response Sample]** をクリックし、今のテスト結果を通常の応答サンプルとして保存します。 機密データが応答に含まれている場合、手動で編集することができます。



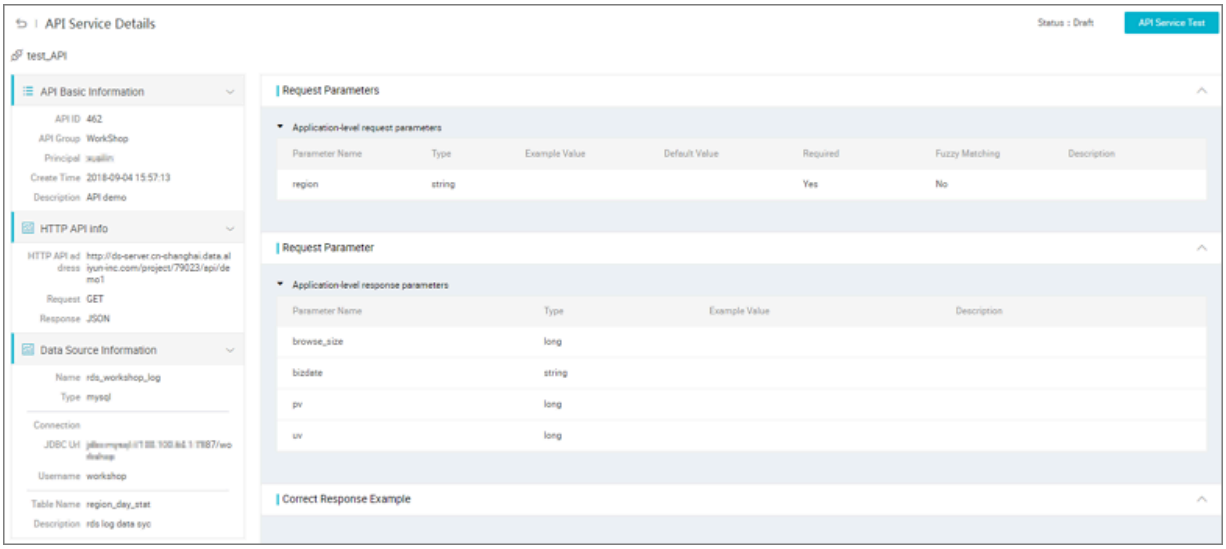
注：

- 標準の応答例は、**API** 呼び出し元に重要な参照値を提供します。 可能であれば例を明記します。
- **API** 呼び出し遅延は、現在の **API** リクエストの遅延です。これは、**API** のパフォーマンスを評価する際に使用されます。 レイテンシが高すぎる場合は、データベースの最適化を検討してください。

API のテストが完了したら、**[Finish]** をクリックします。 以上により、**API** は正常に作成されます。

API 詳細の閲覧

[API Service list] ページに戻り、操作列の **[details]** をクリックし、**API** の詳細を表示します。 このページには、呼び出し元から見た **API** に関する詳細情報が表示されます。



2.3.4 スクリプトモードでの API の生成

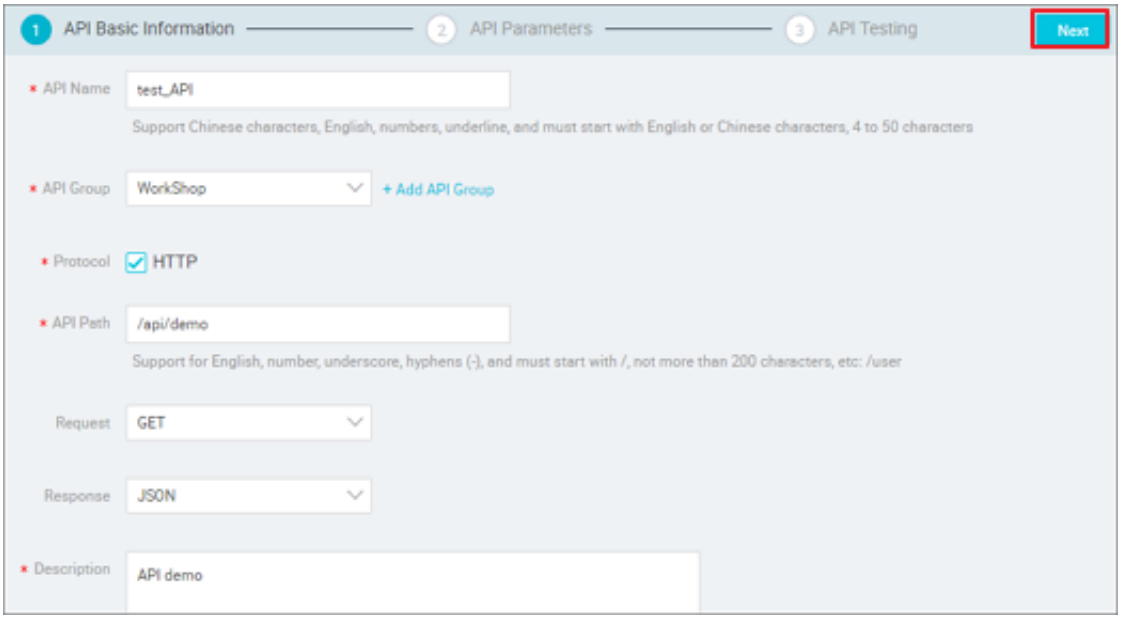
本ページでは、スクリプトモードでの **API** 生成の実行手順を説明します。

パーソナライズされたクエリに対するハイエンドユーザーの要件を満たすため、データサービスでは **SQL** をカスタマイズするためのスクリプトパターンも提供しています。これにより、**API** の**SQL** クエリを自分自身で作成することが可能になります。複数テーブルの関連付け、複雑なクエリ条件、および集約関数に対応しています。

API 基本情報の設定

1. **[API Service list]** > **[Generate API]** に移動します。

2. **[Script Mode]** をクリックし、**API** の基本事項を入力します。



設定では、 **API** グループ化の設定に注意してください。 **API** グループには、特定のシナリオに使用される各種 **API** 集が含まれています。 これは **API Gateway** での最小管理単位です。 **Alibaba Cloud API** マーケットでは、各 **API** グループは特定の **API** に対応しています。

> 注：
> **API** グループ化の設定例は次のとおりです。
>
> たとえば、天気を照会するのに **API** プロダクトの設定を行いたい場合、都市名検索による天気検索 **API**、観光地天気検索 **API**、郵便番号検索による天気 **API** の **3** 種類の **API** プロダクトを設定し、"天気の照会" と名付けた **API** グループを作成し、このグループに上記の **3** つの **API** を配置します。 その **API** がマーケットに公開されると、天気の照会プロダクトとして表示されます。
>
> もちろん、ユーザーが生成した **API** がユーザー自身のアプリ内で使用される場合、分類するのにグループ化を行うことができます。

現在、ビルド **API** は **HTTP** プロトコル、**GET** リクエストモード、および **JSON** リターンのみをサポートしています。

3. **API** の基本情報を入力したら、**[Next]** をクリックし、**API** パラメーター設定ページに移動します。

API パラメーターの設定

1. データソースとテーブルを選択します。

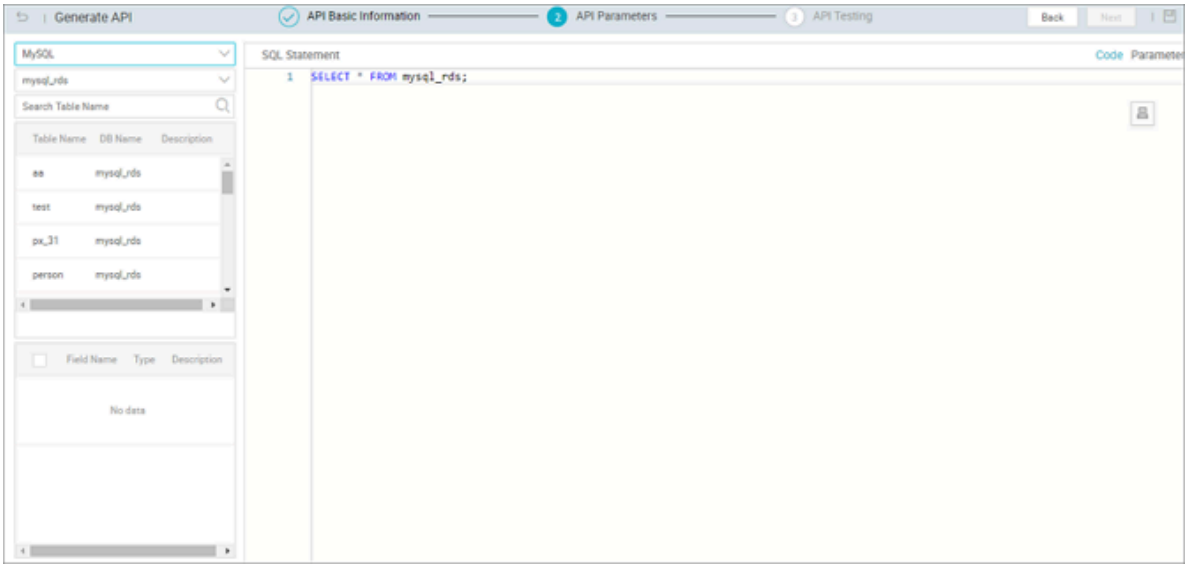
   [Data source type] > **[Data source name]** > **[Data Table]** に移動します。データテーブルリスト内の適切なテーブル名をクリックし、このテーブルのフィールド情報を表示します。

   注：

   - データセット内でデータソースを事前に設定する必要があります。
   - データソースを選択する必要があります。 データソース間のテーブル結合クエリには対応していません。

2. **API** の **SQL** クエリを書きます。

   右側のコードボックスに **SQL** コードを入力することができます。 システムは、フィールドのリスト中のフィールドをチェックするワンクリックの **SQL** 機能に対応しています。**[Generate SQL]** をクリックすると、`SELECT xxx FROM xxx` の **SQL** 文が自動的に生成され、右カーソルに挿入されます。

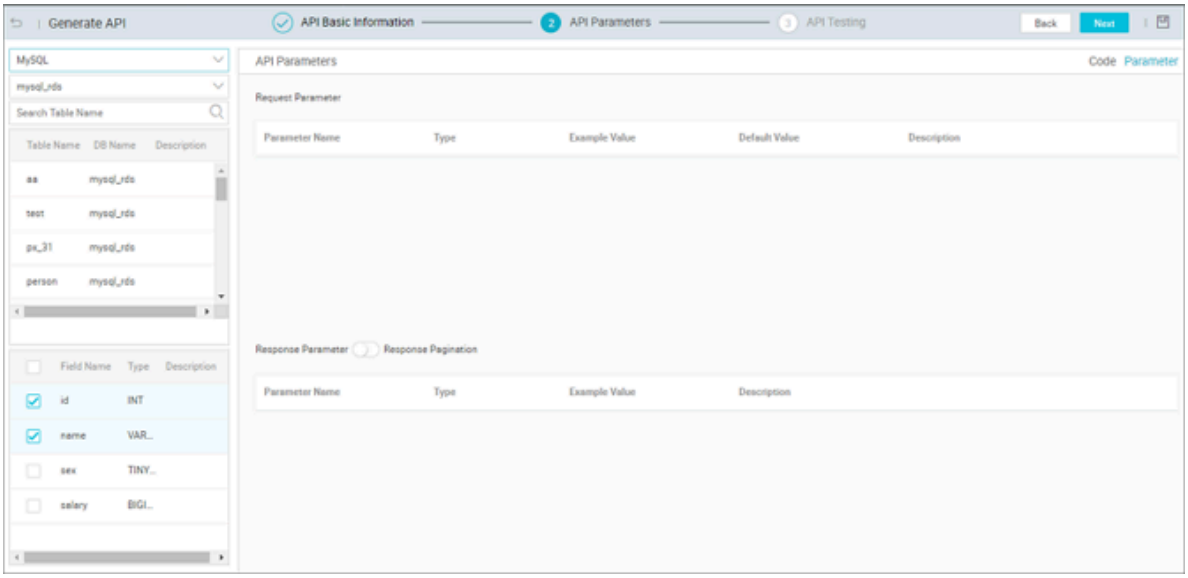
   注：

   - ワンクリックの **SQL** 追加は、フィールド数が比較的多い場合に特に便利です。これにより、**SQL** 作成の効率が大幅に向上します。
   - **SELECT** クエリのフィールドは、**API** のリターンパラメーターです。**where** 条件のパラメーターは、**API** のリクエストパラメーターです。リクエストパラメーターは **"$"** で識別されます。

3. 最後に、パラメーター情報を編集して完了です。

   API クエリの **SQL** を作成した後、右上の **[parameters]**をクリックし、パラメーター情報の編集ページに切り替えます。編集ページでは、タイプ、サンプル値、既定値、およびパラメーターの説明を編集します。タイプとパラメーターの説明の編集は必須です。

   > 注：
   > **API** の呼び出し者が **API** をより総合的に理解できるようにするため、可能な限り、**API** パラメーター情報を入力します。



設定プロセスでは、ページング結果の戻り設定に注意を払う必要があります。

- **[Response pagination]** を有効にしない場合、既定では、**API** は最大 **500** レコードを出力します。
- 返される結果が **500** を超える可能性がある場合、**[Response pagination]** 機能をオンにします。

次のパブリックパラメーターは、**[Response pagination]** 機能が有効になっている場合にのみ、使用することができます。

- 共通のリクエストパラメーター
  - **pageNum:** 現在のページ番号です。
  - **Pagesize:** ページサイズです。 つまり、**1** ページごとのレコード数です。

- 共通のレスポンスパラメーター
  - **pageNum:** 現在のページ番号です。
  - **Pagesize:** ページサイズです。 つまり、**1** ページごとのレコード数です。
  - **totalNum:** レコードの合計数です。

 注 :

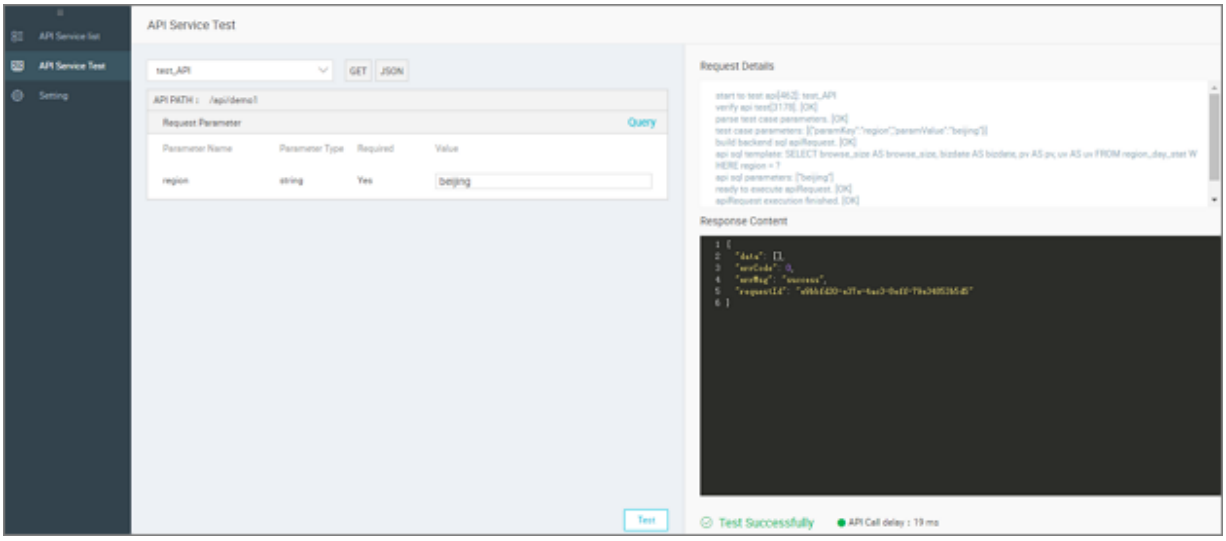
SQL ルールプロンプト

- 対応している **SQL** 文は **1** つだけであり、複数の **SQL** 文には対応していません。
- 対応しているのは、**SELECT** クラスだけです。 **INSERT**、**UPDATE**、**DELETE** などの他のクラスは対応していません。
- **SELECT** クラスを対象としたクエリフィールドは、**API** のリターンパラメーターです。 **where** 条件内の **$ {Param}** の変数 **"Param"** は、**API** のリクエストパラメーターです。
- `SELECT \*` はサポートされていません。クエリの列は明示的に指定する必要があります。
- 単一のテーブルクエリ、テーブル結合クエリ、および単一のデータソース内のネストされたクエリがサポートされています。
- **SELECT** クエリ列の列名に、テーブル名のプレフィクス (たとえば、**T. name** など) がある場合、このエイリアスをリターンパラメーター名 (たとえば、**T. name** を名前とする) と見なす必要があります。
- 集計関数 (**min / max / sum / count**など) を使用する場合は、エイリアスをリターンパラメーター名として使用する必要があります (**total \ _ num** を **sum (Num)** とする)。
- **SQL** では、リクエストパラメーターが置き換えられるとき、**$ {Param}** は一様で、文字列 **}** に **$ {Param}** を含んでいます。 **$ {Param}** にエスケープ文字 **** があるとき、通常の文字列として処理されるリクエストパラメーター処理を行いません。
- たとえば、**'$ {ID}'**、 **'ABC $ {xyz} 123'** のように、 **$ {Param}** を引用符で囲むことはサポートされていません。 **'abc ', '$ {xyz}', '123'** の文字列の結合については、必要であれば実装することができます。

API パラメーターの設定が完了したら、**[Next]** をクリックし、**API** のテストセクションに移動します。

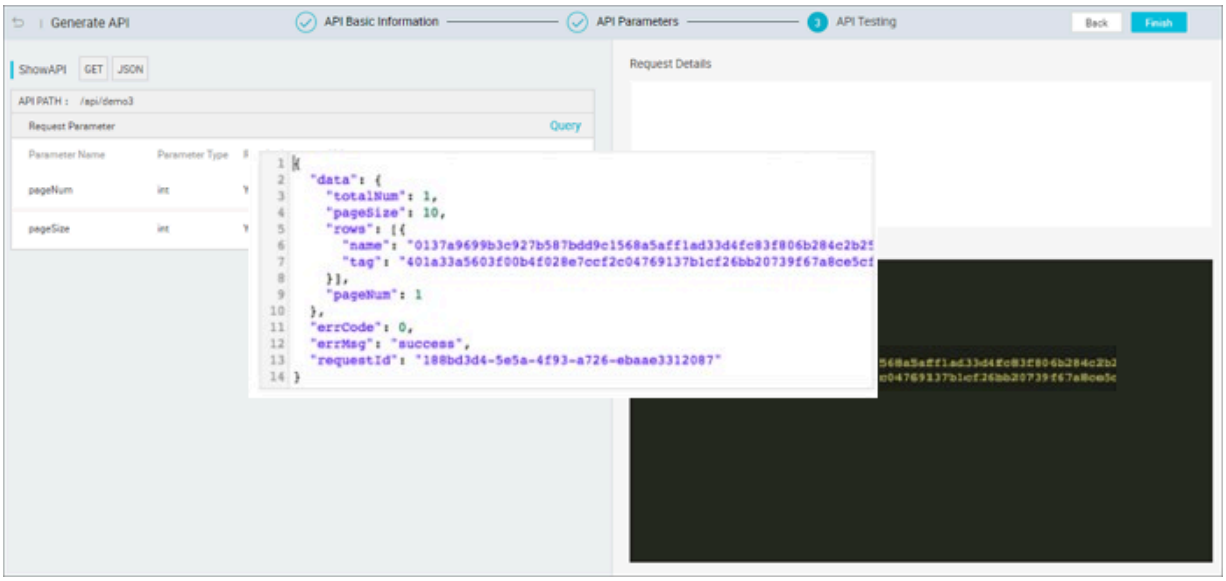
API のテスト

API パラメーターの設定が完了したら、**API** のテストを開始します。



パラメーターを設定し、**[Start Test]** をクリックして、**API** リクエストをオンラインで送信します。 **API** リクエストの詳細と応答結果が右側に表示されます。 テストが失敗した場合は、エラーメッセージを注意深く読み、適切な調整を行って **API** を再テストします。

設定プロセスでは、標準の応答例の設定に注意する必要があります。 **API** をテストするとき、システムは自動的にエラー例およびエラーコードを生成します。 ただし、標準の応答例は自動的には生成されません。 テストが成功したら、**[Save as Normal Response Sample]** をクリックし、今のテスト結果を通常の応答サンプルとして保存します。 機密データが応答に含まれている場合、手動で編集することができます。



注：

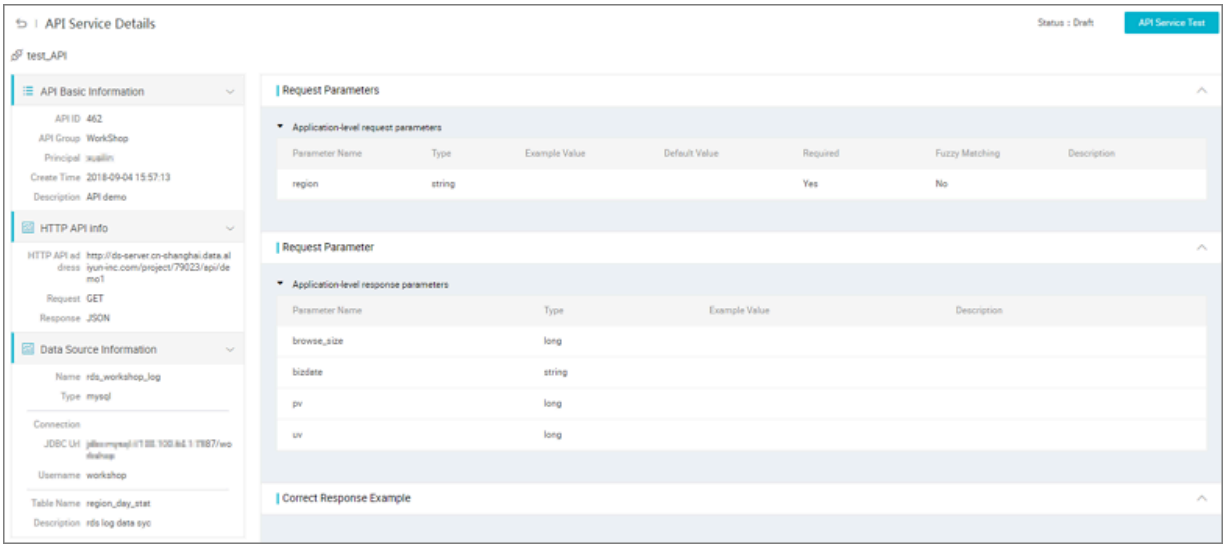
- 標準の応答例は、**API** 呼び出し者に重要な参照値を提供します。 可能であれば例を指定します。

- **API** 呼び出し遅延は、現在の **API** リクエストの遅延です。これは、**API** のパフォーマンスを評価する際に使用されます。 レイテンシが高すぎる場合は、データベースの最適化を検討してください。

API のテストが完了したら、**[Finish]** をクリックします。 これで、データ **API** は正常に作成されました。

API 詳細の閲覧

[API Service list] ページに戻り、操作列の **[details]** をクリックし、**API** の詳細を表示します。 このページには、呼び出し者から見た **API** に関する詳細情報が表示されます。



2.4 API の公開

API Gateway は、完全なライフサイクル管理を提供する **API** ホスティングサービスですライフサイクル管理では、**API** のリリース、管理、**O&M**、販売をカバーしています。 **API Gateway** では、パートナーや開発者を対象として、マイクロサービスの統合、フロントエンドとバックエンドの分離、およびシステム統合を実装するための、シンプルかつ高速で低コスト低リスクのメソッドを提供します。

API Gateway では、権限管理、トラフィック制御、アクセス制御、およびメータリングサービスを提供します。 これらのサービスにより、**API** の作成、モニタリング、保全を簡単に行うことができます。 そのため、データサービスで作成および登録された **API** を **API Gateway** に公開することを推奨します。 データサービスと **API Gateway** は接続されているため、各種 **API** を **API Gateway** に簡単に公開することができます。

各種 API の API Gateway への公開

> 注：
> API をリリースするには、まず API Gateway サービスを有効にする必要があります。

API Gateway を有効にした後、[API Service list] の操作列で、[Publish] をクリックし、API を API Gateway にリリースします。 システムは、パブリッシュプロセス中に、API を API Gateway に自動的に登録します。 システムは、API グループと同じ名前でグループを API Gateway に作成し、そのグループに API をリリースします。

リリース後、API Gateway コンソールにアクセスし、API 情報を表示します。 API Gateway では、スロットルおよびアクセス制御機能を設定することもできます。

API をアプリケーションから呼び出す必要がある場合、API Gateway でアプリケーションを作成し、そのアプリケーションに対して API を許可し、AppKey と AppSecret を使用して、署名の呼び出しを暗号化する必要があります。 詳しくは、「*API Gateway help documentation*」をご参照ください。 同時に、API Gateway では主流のプログラミング言語で開発ができる SDK の提供もしており、自分のアプリケーションに対して、自分が作成した API をすばやく統合することができます。詳しくは、「*SDK のダウンロードとユーザーガイド*」をご参照ください。

各種 API の Alibaba Cloud API マーケットプレイスへの公開

ユーザーの各種 API を、データサービスから API Gateway にパブリッシュした後、それらを Alibaba Cloud API Marketplace に公開することができます。 この方法により、ユーザーの会社は経済的利益を簡単に得ることができます。

API を Ali cloud の API マーケットに販売する前に、まず最初に、サービスプロバイダーとして Ali cloud の API market にログインする必要があります。

> 注：
> 次の図に示すように、API Marketplace への移動を選択します。 注記: Alibaba Cloud API Marketplaceに参加することができるのは、エンタープライズユーザーのみになります。

手順

1. Ali cloud の「サービスプロバイダープラットフォーム」に移動します。
2. [commodity management] > [publish the merchandise] をクリックし、API サービスとしてアクセスタイプをクリックします。
3. 一覧表示したい API グループを選択します (1 つのグループは 1 つの API 商品に対応します)。
4. 商品情報を設定して、監査を送信します。

製品が **Alibaba Cloud API Marketplace** に正常に公開されると、世界中のユーザーがその製品を購入することができるようになります。

2.5 API の削除

API を削除するには、**[API Service list]** の操作列で、**[More]** > **[Delete]** をクリックします。

注:

- **API** は、オフラインステータスのときにのみ削除できます。 オンラインの場合は、**API** を非推奨にしてから削除します。
- 削除操作は取り消しができません。 **API** の削除は、注意して行ってください。

2.6 API の呼び出し

本ページでは、**API** が **API Gateway** でリリースされた後、**API** を呼び出す方法について説明します。

API Gateway では、**API** の権限付与および **API** の呼び出しに使用する **SDK** を提供します。ユーザー自身、ユーザーの同僚、または第三者に対して、**API** を使用するための権限付与を行うことができます。 **API** を呼び出したい場合は、以下の操作を行います。

API を呼び出すための 3 つの要素

API を呼び出すには、次の **3** つの要素が必要です。

- **API:** ユーザーが呼び出そうとしている **API** です。**API** パラメーターによって明確に定義されています。
- アプリ**: API** を呼び出すために使用するアイデンティテです。 **AppKey** と **AppSecret** は、ユーザーのアイデンティティを認証するために提供されています。
- **API** とアプリの権限関係**:** アプリが **API** を呼び出す必要がある場合、アプリは呼び出そうとしている **API** へのアクセス許可を持っている必要があります 権限付与により、**API** へのアクセス許可が与えられます。

手順

1. **API** ドキュメントの取得

   取得方法は、**API** の取得に使用したチャンネルによって異なります。 通常、**API** ドキュメントは、データマーケットから購入した **API** サービスに分類され、購入する必要はありません。**2** つの取得方法が、プロバイダーにより積極的に指定されています。 詳しくは、「*get API documentation*」をご参照ください。

2. プロジェクトの作成

   アプリは、**API** を呼び出すために使用するアイデンティティです。 各アプリには、**AppKey** と **AppSecret** のセットがあり、これらはアカウントとパスワードに相当します。 詳しくは、「アプリケーションの作成」をご参照ください。

3. 権限の取得

   権限付与とは、アプリに **API** を呼び出す許可を与えることを意味します。 **API** を呼び出すためには、まず、ユーザーのアプリで権限付与が行われる必要があります。

   権限付与の方法は、**API** の入手に使用するチャンネルによって異なります。 詳しくは、「許可の取得」をご参照ください。

4. **API** の呼び出し

   API の呼び出し方法は、**API Gateway** コンソールで提供されている多言語呼び出しのサンプルを直接使用する方法、もしくは自己コンパイルされた **HTTP** または **HTTPS** リクエストを使用する方法があります。 詳しくは、「*APIの呼び出し*」をご参照ください。

2.7 よくある質問

- **Q: API Gateway** は有効にする必要がありますか。

  A: API Gateway は **API** ホスティングサービスを提供しています。 自分の **API** を他のユーザーに公開する予定の場合は、まず、**API Gateway** サービスを有効にする必要があります。

- **Q:** データソースはどこで設定が行えますか。

  A: データソースを作成するには、**[DataWorks]**、**[Data Integration]**、**[Data Sources]** と、順にクリックします。 設定後、**Data Service** はデータソース情報を自動的に読み取ります。

- **Q:** ウィザードで作成される **API** と、スクリプトで作成される **API** の違いは何ですか。

  A: スクリプトモードは、より強力な機能を提供します。 詳しくは、「スクリプトモードでの *API* の生成」をご参照ください。

- **Q: Data Service** の **API** グループとはどういったものですか。 **API Gateway** における **API** グループと同様ですか。

  A: 特定のシナリオにおいて、ある **API** グループには複数の **API** が含まれています。 **API** グループ は最小単位です。 一言で言えば、**2** つは同等です。 **Data Service** から **API Gateway** に **API** グループを公開すると、ゲートウェイは自動的に同じ名前の **API** グループを作成します。
- **Q: API** グループを適切に設定する方法を教えてください。

  A: 通常、ある **API** グループには、同様の機能を提供する **API** や、特定の問題を解決する **API** が含まれています。 たとえば、都市名で天気を照会する **API** と、緯度と経度で天気を照会する **API** は、**API**グループ名を **"weather query"** とした **API**グループに入れます。
- **Q: API** グループは何個まで作成することができますか。

  A: Alibaba Cloud のアカウントでは、最大 **100** 個の **API** グループを作成することができます。
- **Q: API** の応答出力のページネーションは、どのような状況で有効にする必要があるのでしょうか。

  A: 既定では、**API** は最大 **500** レコードを出力します。 **API** の応答出力のページネーションを有効にすると、さらに多くのレコードを出力することができます。 **API** のリクエストパラメーターが設定されていない場合、**API** は大量のレコードを出力することがあります。その場合、**API** の応答出力ページネーションは、自動的に有効になります。
- **Q:** データソースで作成される **API** は、**POST** リクエストに対応していますか。

  A: 現在、作成される **API** は **GET** リクエストにのみ、対応しています。
- **Q: Data Service** は **HTTP** に対応していますか。

  A: 現在、**Data Service** は **HTTP** に対応していません。 **HTTP** は以降のバージョンで、対応となる可能性があります。