**语义模型扩展方案**

By 边传猛

目录

[1 语义函数扩展 2](#_Toc418610914)

[1.1 接口 2](#_Toc418610915)

[1.2 实现 3](#_Toc418610916)

[1.3 配置文件 4](#_Toc418610917)

[2 脚本规则扩展 5](#_Toc418610918)

[2.1 接口 6](#_Toc418610919)

[2.2 实现 7](#_Toc418610922)

[2.3 配置文件 7](#_Toc418610923)

[2.4 使用 8](#_Toc418610924)

[3 语义提供者扩展 9](#_Toc418610925)

[3.1 接口 9](#_Toc418610927)

[3.2 实现 11](#_Toc418610928)

[3.3 配置文件 13](#_Toc418610929)

[3.4 使用 14](#_Toc418610930)

[4 数据加工使用 15](#_Toc418610931)

[4.1 概念 15](#_Toc418610933)

[4.2 定位 15](#_Toc418610934)

[4.3 执行原理 16](#_Toc418610935)

[4.4 使用 16](#_Toc418610936)

[4.5 常见问题 18](#_Toc418610937)

[5 元定义驱动扩展 18](#_Toc418610938)

[5.1 接口 19](#_Toc418610940)

[5.2 实现 20](#_Toc418610941)

[5.3 配置文件 20](#_Toc418610942)

[5.4 使用 21](#_Toc418610943)

# 语义函数扩展

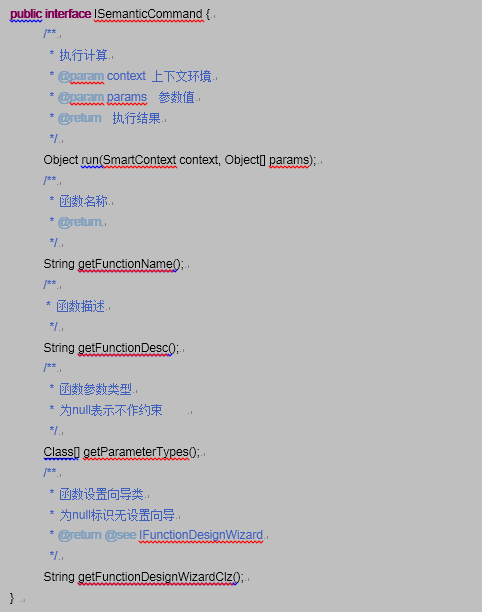
语义模型支持对其函数进行扩展，用户可以根据业务需求扩展自定义函数。自定义函数和预置函数在使用上相同，可用于字段表达式、语义脚本、筛选表达式、关联表达式等处。

## 接口

根据函数接口(nc.pub.smart.script.function.ISemanticCommand)实现自定义函数。类图示意如下：



接口中API如下：



## 实现

对于一般函数，可直接实现nc.pub.smart.script.function.ISemanticCommand。对于数据库类型函数，可继承nc.pub.smart.script.function.DBTypeCommand，实现以下三个接口方法：

**public** **abstract** String getMSSQLValue(Object[] params);

**public** **abstract** String getOrclValue(Object[] params);

**public** **abstract** String getDB2Value(Object[] params);

## 配置文件

默认配置文件路径为${NC\_Home}/resources/smart/function\_default.xml。

为方便各模块扩展函数，在开发环境下，配置文件存放位置为各模块resources下的smart目录：

${NC\_Project}/resources/smart/function\_${模块号}.xml

如总帐项目扩展配置文件路径为：

gl/resources/smart/function\_gl.xml

这样在安装盘中，各模块的扩展配置文件将都在NC\_HOME/resources/smart/目录中。

配置文件示例如下：

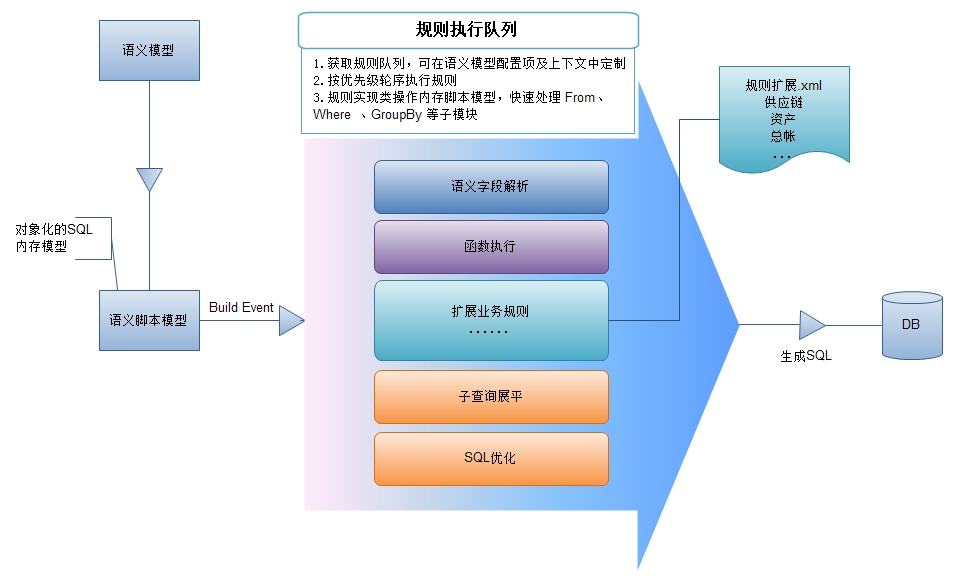
配置文件中是根据驱动名称进行组织的，驱动总共分为两类SmartFunctionDriver和DBFunctionDriver。之所以对驱动进行区分是因为在不同的功能点，加载的函数驱动可能不同。

在实际配置过程中，用户可以根据自己的需要把自定义的函数类全路径名放在已有的驱动定义或新增加自定义的驱动定义中，即可完成函数的扩展。



# 脚本规则扩展

脚本规则是指针对语义脚本的处理逻辑，具体处理过程如下图所示：



脚本规则支持扩展，业务部门可以通过扩展规则、直接操作内存脚本模型，藉此实现特定功能，例如：复杂权限控制等。

## 接口

规则的统一接口为：nc.pub.smart.script.engine.rule.IScriptRule

主要接口方法为：

***/\*\****

***\* 执行规则***

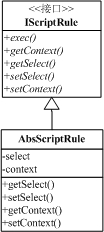
***\*/***

***public Select exec() throws Exception;***

在该接口中，我们可以操作传过来的脚本模型Select各部分，包括select字段列表、from、where、orderBy、groupBy等。修改完脚本模型，把其返回即可。



## 实现



我们可以继承AbsScriptRule来实现自定义的规则。

## 配置文件

编写完规则实现类，我们还需在配置文件中注册该规则，以便在语义模型能加载到。所谓注册，即在配置文件中增加相应配置项。

为方便各模块扩展规则，在开发环境下，配置文件存放位置为各模块resources下的smart目录：

${NC\_Project}/resources/smart/smart\_rule\_${模块号}.xml

如总帐项目扩展配置文件路径为：

gl/resources/smart/smart\_rule\_gl.xml

这样在安装盘中，各模块扩的扩展配置文件将都在NC\_HOME/resources/smart/目录中。

具体配置信息如下：

每个规则注册项包含三种属性：显示名、规则class全路径名、优先级。其格式为：

<nc.pub.smart.script.engine.rule.config.ScriptRuleConfig>

<items>

<nc.pub.smart.script.engine.rule.config.ScriptRuleConfigItem>

<displayName></displayName>

<ruleClz></ruleClz>

<priority></priority>

</nc.pub.smart.script.engine.rule.config.ScriptRuleConfigItem>

</items>

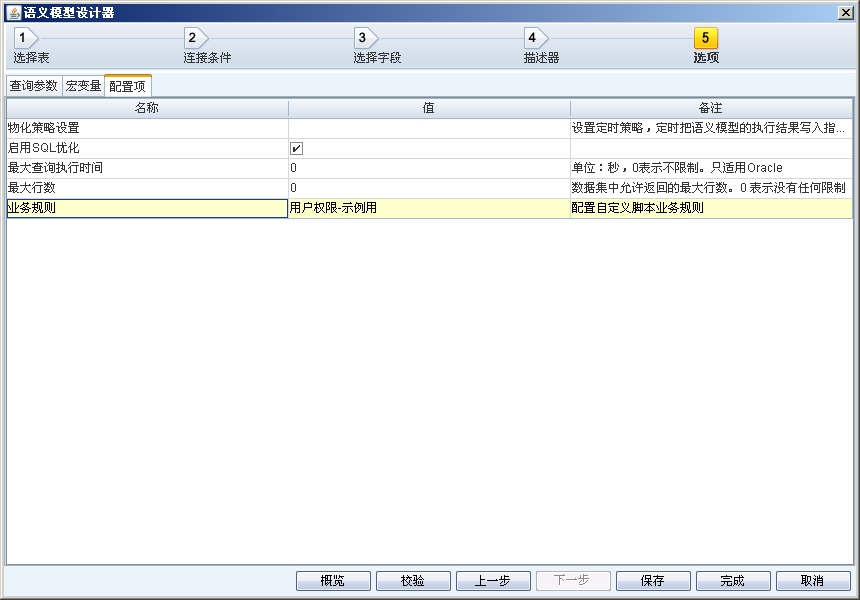
</nc.pub.smart.script.engine.rule.config.ScriptRuleConfig>

其中，priority优先级是用于确定各规则的调用顺序***。***

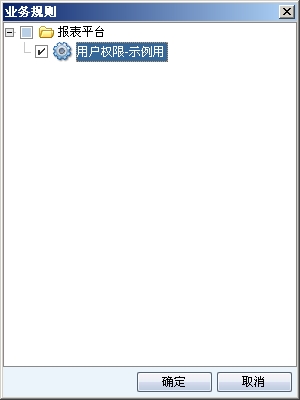
至此，我们的规则扩展就全部实现了，在语义模型的设计向导中就会支持该规则。

## 使用

打开 语义模型设计向导--最后一步“选项”--配置项，“业务规则”配置项，



设置值时弹出规则选择对话框：



勾选所需规则，支持多选。这些选中规则即可在脚本引擎解析时调用，来对语义脚本进行处理。

# 语义提供者扩展

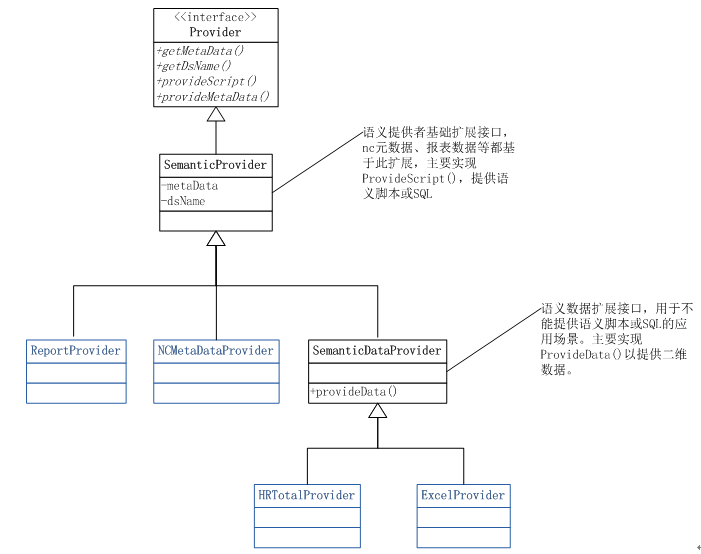
语义提供者，表述了一类取数方式，或者说如何提供数据的方式。在语义模型中，语义提供者负责把一类业务取数过程以语义脚本的形式描述出来。



## 接口

语义提供者包括NC元数据、DW元数据、以及语义脚本和业务代码扩展提供者。

提供总帐、HR、供应链、报表等业务数据扩展。其整个体系结构可由下图表示：



其中，Provider是语义提供者的接口；

SemanticProvider是基础扩展抽象类，对能把取数过程以脚本形式描述的语义提供者可继承此类；

SemanticDataProvider是语义数据扩展抽象类，对不能以脚本形式描述取数过程，只能提供二维数据的提供者，可继承此类。

SemanticSqlProvider适用于提供者在运行时根据执行环境context返回不同取数sql

与SqlProvider的区别在于：

SqlProvider的sql结构在定义态已经确定；

SemanticSqlProvider是在运行时，经过一系列业务处理，返回最终取数sql.

上述图中，蓝色代表具体实现类。

通过以上的介绍我们可以得知，Provider定义了语义提供者的接口规范，SemanticProvider、SemanticDataProvider、SemanticSqlProvider则是我们具体实现提供者时要继承的抽象类。现对这四个类的主要接口做重点介绍。

* Provider

|  |  |
| --- | --- |
| **接口方法** | **方法说明** |
| String getCode() | 获取编码。该编码用于标示语义提供者 |
| MetaData getMetaData() | 获取元数据。元数据中包含了该提供者提供的数据的描述信息，例如：字段名、字段数据类型、字段精度 等 |
| String getTitle() | 获取标题，即该提供者的显示名称 |
| MetaData  provideMetaData(SmartContext context) | 构造元数据。  入参参数context是提供者构造元数据时的执行环境。  此方法通常在设计态使用，用于重新构造元数据。 |
| String provideScript(SmartContext context) | 构造语义脚本。  入参参数context是提供者构造脚本时的执行环境。  此方法通常在运行态使用，用于把业务取数过程转换为语义脚本。 |
| void setCode(String code) | 设置编码 |
| void setMetaData(MetaData metaData) | 设置元数据。  此方法通常在设计态使用，一般是先调用provideMetaData(context)构造元数据，然后调用此方法来把元数据保存在语义提供者。 |
| void setTitle(String title) | 设置标题，即显示名 |

* SemanticProvider

|  |  |
| --- | --- |
| **接口方法** | **方法说明** |
| MetaData  provideMetaData(SmartContext context) | 同Provider  对于继承此类的提供者，必须实现此方法。 |
| String provideScript(SmartContext context) | 同Provider  对于继承此类的提供者，必须实现此方法。 |

* SemanticDataProvider

|  |  |
| --- | --- |
| **接口方法** | **方法说明** |
| DataSet provideData(SmartContext context) | 提供二维数据。  入参context提供执行环境；  返回结果数据以DataSet形式展现。DataSet主要包含两部分：二维数据数组、元数据。  对于继承此类的提供者，必须实现此方法。 |

* SemanticSqlProvider

|  |  |
| --- | --- |
| **接口方法** | **方法说明** |
| String provideSql (SmartContext context) | 提供sql。  入参context提供执行环境；  返回结果数据以sql形式展现。  对于继承此类的提供者，必须实现此方法。 |

## 实现

前面介绍了语义提供者的整个体系结构，现在我们拿一个具体例子来讲解如何实现一个语义提供者。

我们以比较简单的“数据表”这类取数方式来做示例。

扩展语义提供者可分为以下三步：

* + 1. **实现语义提供者类**

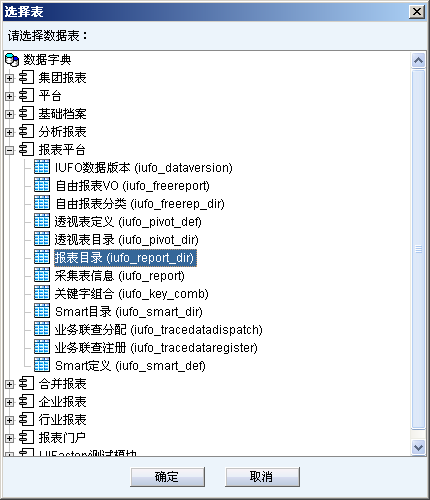
由前文介绍，我们知道实现一个提供者有三种方式：

* + - 1. **继承SemanticProvider：**能把取数过程以脚本形式描述的语义提供者可继承此类

我们现以数据表提供者为例来讲解如何以此种方式实现语义提供者。

数据表提供者对应的实现类为：DbTableProvider，该类继承于SemanticProvider，实现了接口provideMetaData(SmartContext context) ， provideScript(SmartContext context)。

数据表提供者，是把NC元数据底层数据模型中的一张表作为操作对象，从中抽取数据。其在语义模型中的操作是这样的：



在上级“模块”目录上选中模块“平台”，展开后在子节点上选中“sm\_user”这张表。

在数据表提供者DbTableProvider中，我们只需要存储一条信息：表名。这些信息可以看做取数过程的业务描述，接下来我们做的就是把这些业务描述转换为以语义模型中的概念来进行描述。

**实现provideMetaData(SmartContext context)接口：**有了表名信息，我们就可以把其列信息拿到，列Column解析为字段Field，每列对应一个字段，多个列对应的多个字段就组合为元数据MetaData。

**实现provideScript(SmartContext context)接口：**数据表提供者比较简单，直接返回表名即可。

到此，我们的数据表语义提供者类就实现完毕。

* + - 1. **继承SemanticDataProvider：**不能以脚本形式描述取数过程，只能提供二维数据的提供者，可继承此类

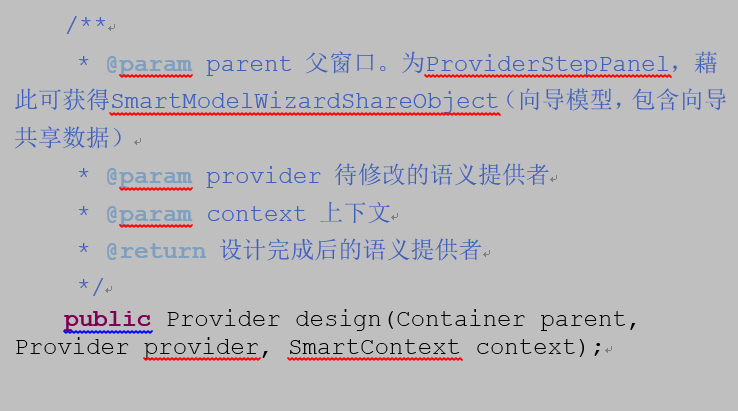
例如，供应链中有些查询并不能直接通过一条sql就能查出，中间可能经过一系列复杂的代码运算逻辑来构造这个结果数据，这时我们就可以继承SemanticDataProvider来实现语义提供者。

继承此类只需要实现provideData(SmartContext context)接口：在该方法中，我们编写运算逻辑，构造最终的结果数据DataSet，返回之即可。

在此我们有必要介绍下DataSet的结构。DataSet主要包含两部分：元数据MetaData，数据容器Object[][]（即二维数组）。Object[][]即是最终的结果数据，MetaData是对数据的描述信息，包含对应字段信息：字段名、数据类型、数据精度 等。

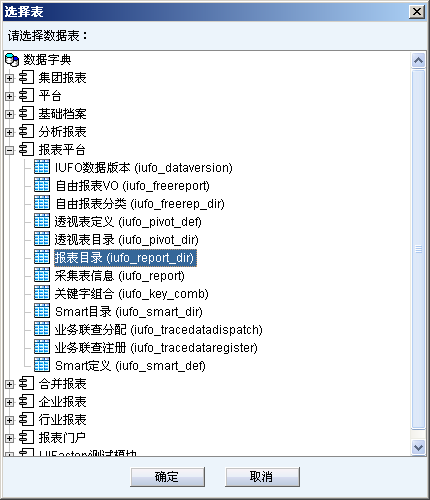
* + - 1. **继承SemanticSqlProvider：**应用方式与**SemanticDataProvider**类似，不同在于，**SemanticDataProvider**最终返回二维数据，而**SemanticSqlProvider**返回sql语句。
    1. **实现语义提供者设计向导类**

有了语义提供者类，我们还需要一个设计器来设计语义提供者。每个设计器都需要实现接口IProviderDesignWizard。该接口只有一个方法：



当设计语义提供者时，会调用此方法，调用完成返回提供者实例。

数据表提供者设计向导类为DbTableProviderDesigner。其界面效果如下图所示：



其具体处理流程为：调用design()方法，打开一如上图所示对话框，用户选中表后，点“确定”按钮，构造一个DbTableProvider实例，返回之。

## 配置文件

编写完语义提供者类、设计向导类，我们还需在配置文件中注册该种语义提供者，以便在语义模型能加载到。所谓注册，即在配置文件中增加相应配置项。

为方便各模块扩展语义提供者，在开发环境下，配置文件存放位置为各模块resources下的smart目录：

${NC\_Project}/resources/smart/smart\_${模块号}.xml

如总帐项目扩展配置文件路径为：

gl/resources/smart/smart\_gl.xml

这样在安装盘中，各模块扩的扩展配置文件将都在NC\_HOME/resources/smart/目录中。

具体配置信息如下：

每个语义提供者注册项包含五种属性：名称、图标路径、类型、语义提供者类名、设计向导类名。其格式为：

<nc.vo.smart.config.ProviderConfig>

<title></title>

<icon></icon>

<type></type>

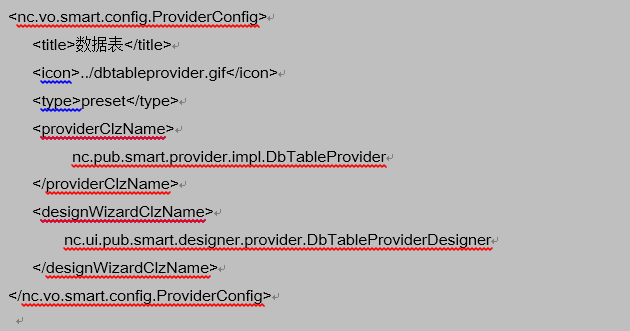
<providerClzName></providerClzName>

<designWizardClzName></designWizardClzName>

</nc.vo.smart.config.ProviderConfig>

其中，type类型是用于区分预置的语义提供者和用户自定义的语义提供者，其有效值有：preset、custom。分别对应系统预置、用户自定义两种类别。***一般用户在自己扩展实现自定义的语义提供者时，可以直接去掉type项，或者把该项设置为custom。***

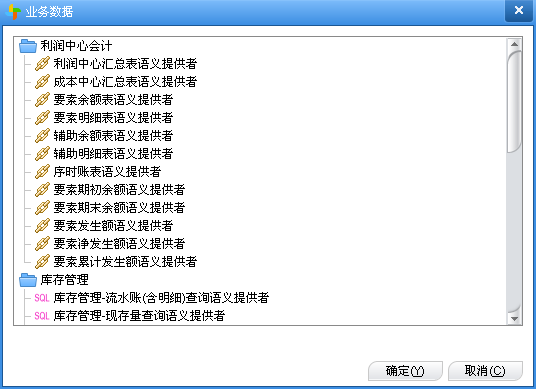
以数据表提供者举例，其配置项为：



至此，我们的语义提供者就全部实现了，在语义模型的设计向导中就会支持该类型的语义提供者。

## 使用

打开语义模型设计向导，第一步“选择表”，点击“业务数据”按钮，就能看到所有系统中注册的语义提供者。选择提供者，点击“确定”按钮后就会打开该提供者的设计器进行业务设计。



# 数据加工使用



## 概念

   数据加工是语义模型提供的一类取数方式，支持用户使用java代码来定义一段取数逻辑。

## 定位

   考虑到在现实应用中业务逻辑的复杂性，单纯依赖sql语句，用户并不是总能得到想要的结果集。

   因此提供了在语义模型嵌入Java代码的功能，用于单纯sql处理不了的场景。

   由于数据加工是在从数据库中取出结果集后在内存中进行操作，考虑到资源和效率问题，我们建议只有在SQL解决不了的数据处理要求，才使用数据加工实现。

   由于在数据加工中提供的通过Java代码实现相关数据处理的功能，理论上讲其可以满足任意相关的业务逻辑需求，复杂的业务逻辑算法需要在实际应用开发中按照不同情况和场景灵活使用。

## 执行原理

   在使用时，数据加工分为两个步骤：

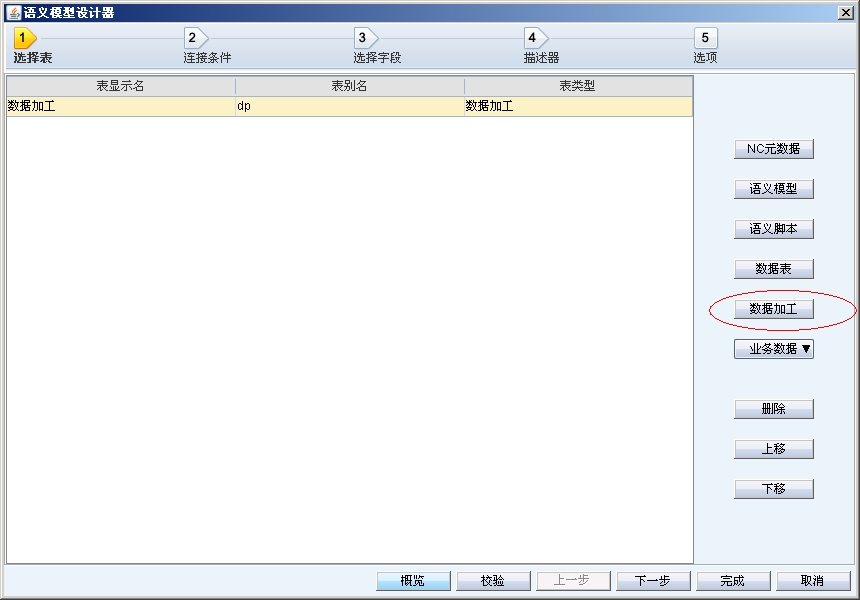
     第一步使用java代码定义取数逻辑，最终返回sql语句 或 结果集DataSet。基于性能考虑，推荐返回值为sql；

     第二步定义元数据，即对上述代码返回的数据的描述，包括：有哪几列、每一列的数据类型、长度、精度等；

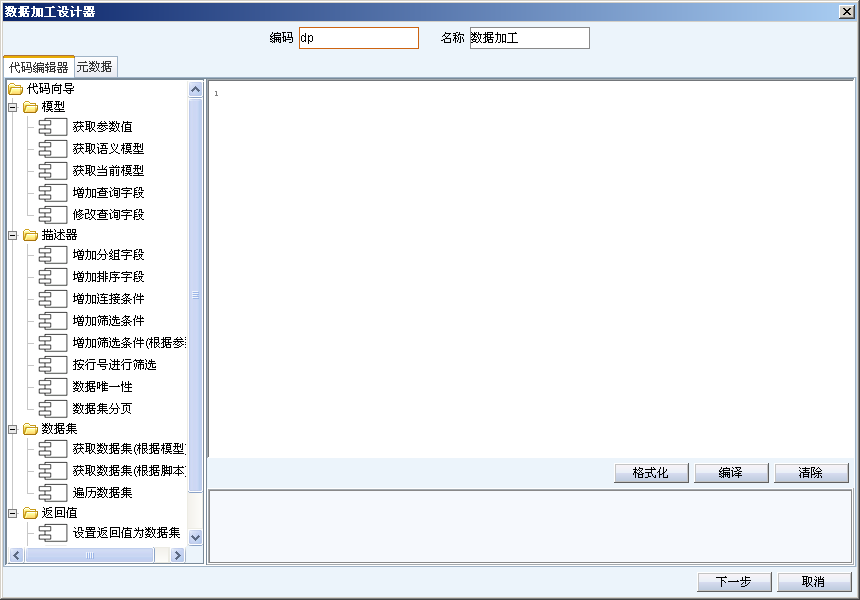
   执行时，上述定义的java代码会在后台被调用，拿到返回的sql语句或结果集DataSet（DataSet需要持久化到数据库中的临时表中），然后和其他逻辑表（provider）进行复合取数。

## 使用

    使用方式如下：

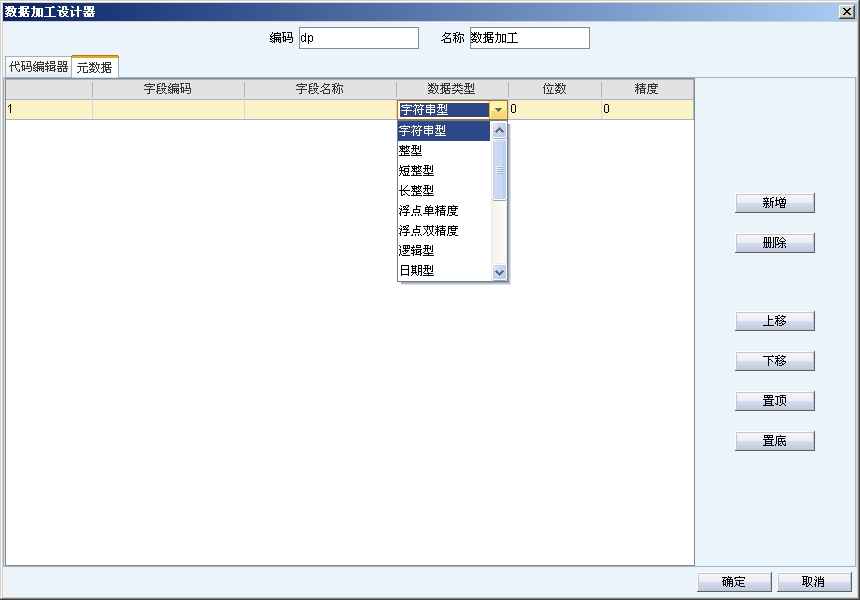


弹出设计界面如下：



在上述编辑器中编辑java代码，左边提供了辅助代码片段生成向导。

步骤“元数据”，是要求代码设计者对代码执行完后的结果数据进行描述，即返回几列数据，每列的列名、数据类型等信息。

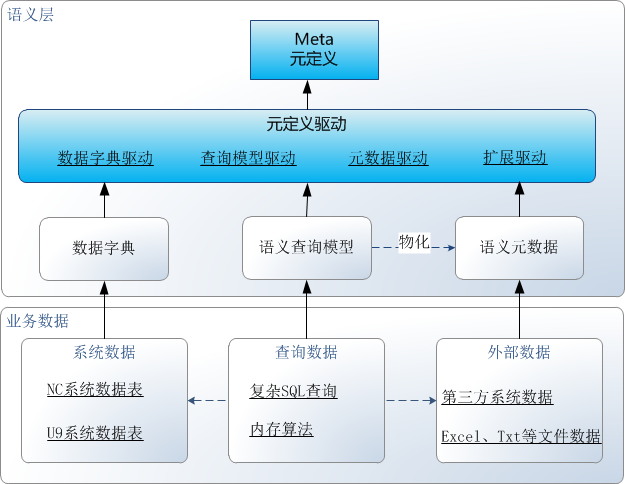


## 常见问题

1. 数据加工java代码中，调用语义模型的类时，直接使用类名，而对业务部门的类，需要使用全路径名。
2. java代码中封装DataSet时，需要设置元数据MetaData（具体参见相关类）。
3. java代码中**通过调用代码setDataSet（DataSet）设置最终需要返回的结果数据集DataSet，通过调用代码setResultSQL(sql)设置sql返回值。**
4. 这部分代码是在后台调用，切忌调用UI类
5. java代码最终生成的类继承于nc.pub.smart.model.code.CodeProcessor ，实现其process() 方法
6. java代码生成的类放在 NC\_Home\modules\bapub\META-INF\var\classes 下

# 元定义驱动扩展

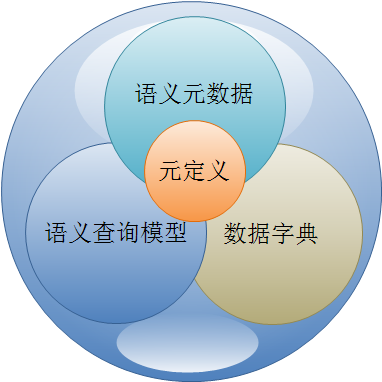
元定义是语义层的核心概念，是对二维业务数据的抽象描述。



元定义通过驱动的方式来挂接不同类型的业务数据，针对NC系统，主要有以下三种：

语义元数据、语义查询模型、数据字典。

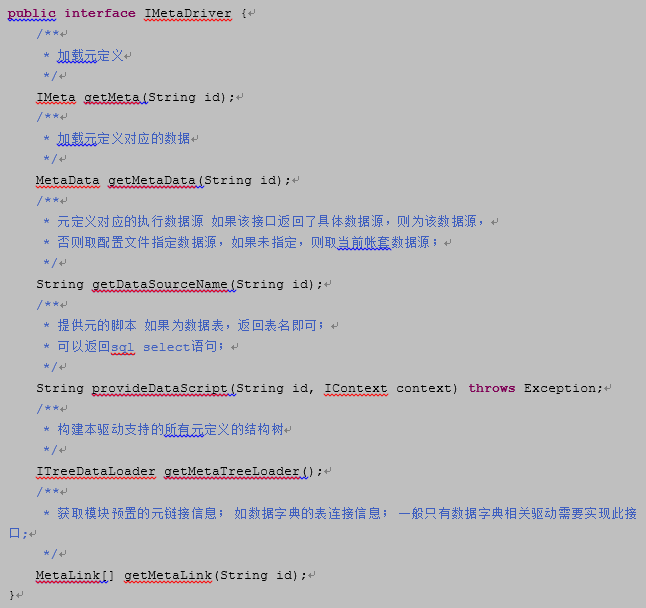
另外，分析平台提供了通用的数据库驱动来支持第三方的数据库。业务部门可以通过扩展驱动的方式来支持自己的业务数据。





## 接口

元定义驱动的接口为nc.bs.bi.meta.IMetaDriver。包含的接口方法有：



## 实现

扩展驱动时直接实现上述接口。可参考现有实现类DbMetaDriver、SmartMetaDriver等。

## 配置文件

元定义驱动配置文件为{NC\_Home}/resources/smart/meta\_driver.xml。配置项如下所示：



也可通过节点：企业建模平台—系统平台—分析平台—元定义驱动 来管理所有的驱动。编辑界面如下图所示：



## 使用

元定义现可用于语义模型、语义关联、分析主题、即时分析 等模块，参照界面如下所示：

