数据源插件编写说明

从DAM7.0开始，所有JDBC类型数据源全部采用插件形式编写，为未来接入更多数据源提供便利。产品组会把6.5支持的所有JDBC类型数据源全部插件化，未来如果项目定开数据源，我们希望项目组能自己开发数据源插件，运行稳定之后可以直接作为产品支持的一部分发布。而且插件化能支持在不更新基础框架的情况下可以更新数据源插件，修复数据源bug。

### 数据源插件代码

gitlab中开辟了一个新的仓库用来存放当前开发完毕的数据源插件，开发新的插件可以以此项目作为模板开发，或者基于此工程开发，仓库地址为：

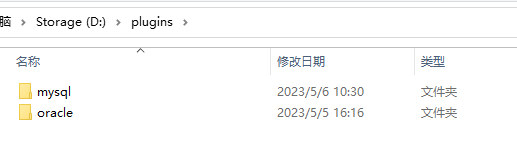
http://192.168.1.34/datablauex/datablau-datasource-plugins

### 数据源插件编写说明

数据源插件是基于maven的一个单独的jar，它需要有一个类实现DAM 7.0中 datablau-datasource-lib 包中 com.datablau.datasource.api.JdbcDatasource接口，另外需要一个插件说明的json文件，名称必须为plugin.json (否则不会加载)。

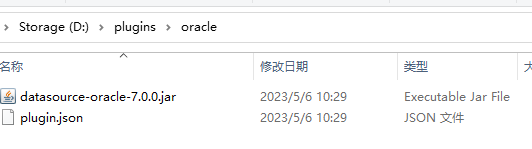
#### 目录结构：

以下为一个样例目录结构：



每一个数据源插件应该放在一个目录之中，所有插件放在一个统一的目录，DatasourceManager会通过配置文件*datablau.plugins.datasource.path = d:/plugins*找到插件所在的目录，然后从中遍历每个文件夹，**每一个文件夹中的jar会被同一个ClassLoader加载**，jar文件不会被递归的加载，所以文件夹中不要再有文件夹。

展开一个目录后，里面结构如图所示：



在此文件夹中包含了两个文件，一个是包含了JdbcDatasource接口实现类的jar文件，另外一个是plugin.json文件用来说明这个插件的关键信息。

在此样例中，我们只有一个jar包，但是我们可以把这个数据源依赖的其它jar包也放在这里，比如jdbc驱动。

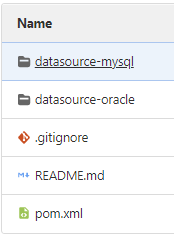
**注意：** *此处的类加载模式不同于tomcat的类加载模式，****此处的类加载器会优先使用父加载器,*** *比如如果当前微服务包含了Foo.class,如果此文件夹的jar中也包含了Foo.class, 那么会优先使用微服务中的Foo.class而不是jar文件中的Foo.class。这种类加载模式对插件开发的影响在接下来会在对应的点描述。*

#### 开始编写一个插件

编写插件，我们推荐直接使用Datablau-Datasource-Plugins这个工程作为母板来开发，按照以下步骤开发：

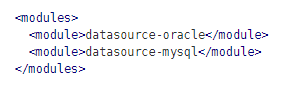
1. 增加插件目录

每个类型数据源是一个目录，命名规范为“datasource-数据源类型”，可以复制一个现有的插件，通过改名增加一个新的类型，比如datasource-flink



1. 添加module到pom.xml

打开根目录下的pom.xml找到<modules>



添加新的module

*<modules>*

*<module>datasource-flink</module>*

*…*

*</modules>*

1. 修改module的pom.xml

module的pom.xml需要修改jar文件和plugin.json的输出目录，主要是maven-jar-plugin的<outputDirectory>路径以及maven-resources-plugsin的<outputDirectory>, 建议放在${basedir}/../plugins/*DATASOURCE\_TYPE*

如果需要其它的依赖包，如果在datablau-dependencies里面已经定义过的，我们建议不定义version，因为按照现在的classloader机制，可能会有版本冲突导致datasource的代码不能被正确加载。

1. 编写插件代码
2. 插件的需求是需要实现接口com.datablau.datasource.api.JdbcDatasource

此接口的方法包含:

*/\*\*  
 \* 获取一个jdbc连接  
 \*/*Connection getConnection();  
  
*/\*\*  
 \* 获取指定数量的jdbc连接  
 \*/*List<Connection> getConnections(int connSize);  
  
*/\*\*  
 \* 获取Database列表  
 \*/*List<String> getDatabases();  
  
*/\*\*  
 \* 获取Schema列表  
 \*/*List<String> getSchemas();  
  
*/\*\*  
 \* 获取驱动的类型名  
 \*/*String getDriverTypeName();  
  
*/\*\*  
 \* 构造表的采样sql，如果page参数空，那么Sql语句不会拼接分页代码  
 \*/*String getTableSampleSql(String schema, String table, QueryPage page, boolean countSql);

*/\*\*  
 \* 数据源的UUID， 如果DDM也需要访问此类型数据源，比如使用DDM里面定义的uniqueId，否则可以自己定义  
 \*/*UUID getUniqueId();  
  
*/\*\*  
 \* 设置数据源的参数  
 \*/*void setProperties(DatasourceProperties properties);  
  
*/\*\*  
 \* 获取数据源的参数  
 \*/*DatasourceProperties getProperties();  
  
*/\*\*  
 \* 检测数据源是否可以连接  
 \* 需要先调用setProperties设置参数后才可以调用此方法  
 \*/*void testConnection() throws ConnectionEstablishException;  
  
*/\*\*  
 \* 获取数据源的类型，必须是全大写字母以及下划线  
 \*/*String getType();  
  
*/\*\*  
 \* 获取当前数据源的所有schemas  
 \*/*List<String> getSchemas();  
  
*/\*\*  
 \* 获取当前数据源的所有databases  
 \*/*List<String> getDatabases();

1. 或者继承基类***com.datablau.datasource.builtin.datasource.DatasourceJdbcBase***

此抽象类实现了大部分的接口，新的插件一般只需要实现以下接口：

***@Override***

***public void testConnection() throws ConnectionEstablishException***

***@Override***

***public String getTableSampleSql(String schema, String table, QueryPage page, boolean countSql)***

***@Override***

***public String getUrl()***

***@Override***

***public String getDriverTypeName()***

***@Override***

***public UUID getUniqueId()***

1. 编写plugin.json

plugin.json是当前插件的描述文件，主要需要包含以下几个属性：

"datasourceType" : "MYSQL", //当前插件对应的数据源类型，大写和下划线  
"displayName" : "MySql", //前端显示的名称，大小写和空格不限制，和数据库官方匹配最好  
"icon" : "mysql", //此plugin对应的图标，前端没有找到对应图标的话会显示默认图标  
"datasourceClass" : "com.datablau.datasources.mysql.MySqlDatasource",

//数据源的class全路径，用来作为当前plugin的接入点  
"jdbcDriverClass" : "com.mysql.jdbc.Driver"

//当前数据源对应的jdbc驱动的class name如果驱动包和plugin放在同一个目录，可

//以不指定驱动管理的驱动，直接使用plugin目录中的驱动

"options" : [{com.datablau.datasource.plugin.PluginOption}…]

com.datablau.datasource.plugin.PluginOption是前端使用的配置项，web端会根据此处的配置显示出相应的数据源页面的输入框，并把用户的输入按照指定的属性值传给后台。

*\* 这里定义的是DatasourceProperties里面parameterMap里面的属性  
 \* 如果属性的值是来自于DatasourceKnownParameterType, 那么其它属性可以不填，前端有默认设置，如果设置了  
 \* 其它属性，那么设置的属性会Override前端默认值。 另外所有属性都会放在DatasourceProperties的parameterMap里面  
 \* 传回给后端，使用code定义的属性名， 比如code = 'FooName', 那么前端会返回属性 'FooName' : 'user input'  
 \*/*@JsonProperty("code")  
private String code;  
@JsonProperty("required")  
private boolean required;  
@JsonProperty("defaultValue")  
private String defaultValue;  
*/\*\*  
 \* 如果code是AuthenticationType, 那么可选值是com.datablau.datasource.plugin.AuthenticationType里面的值  
 \* 如果code是ConnectMode, 那么可选值是com.datablau.datasource.plugin.ConnectMode里面的值*

*\* 其它的可选属性可以自己按需填写  
 \*/*@JsonProperty("candidate")  
private List<String> candidates;  
*/\*\*  
 \* 如果candidates不是空，那么此属性生效，否则此属性无效。  
 \* 如果多选的话，返回值会使用 ';' 号分隔  
 \*/*@JsonProperty("multiple")  
private boolean multipleChoice;  
*/\*\*  
 \* 这个是前端的显示值，如果这个值不设置，那么就走前端默认逻辑，如果设置的了会覆盖前端控件的label的值  
 \*/*@JsonProperty("label")  
private String label;

#### 插件使用

对于7.0来说，所有的数据源管理都在Datablau-Base微服务中管理，所以数据源的管理都在Datablau-Base微服务中完成，其它的微服务都使用Datablau-Base微服务注册的数据源，并且都需要依赖数据源核心包

<dependency>  
 <groupId>com.datablau</groupId>  
 <artifactId>datablau-datasource-lib</artifactId>  
 <version>7.0.0-SNAPSHOT</version>  
</dependency>

此核心包会自动注册数据源相关的bean (scanPackage需要添加下面类的package)

***com.datablau.datasource.util.DatasourceManager***

***com.datablau.datasource.util.DataAccessCustomDriverManager***

另外需要一个本地类实现接口

***com.datablau.datasource.api.JdbcDriverLoader***

(可以直接复制Datablau-Base的com.datablau.base.server.impl.JdbcDriverLoaderImpl)

以及通过HttpInvoker引入base微服务的

***com.datablau.base.api.DatasourceService***

***com.datablau.base.api.DriverService***

***com.datablau.base.api.FileService***

增加配置文件

*datablau.plugins.datasource.path = d:/plugins*

把微服务所需要的数据源插件都放在此目录中。