目录

1 db2text

1.1 概述

这个工具主要用于读取数据库里的对象如表、视图等的结构，根据这些结构生成文本文件。

可以用这些文本文件以及版本控制软件来备份、检查数据库结构的变动情况。还可以使用模板技术根据这些结构自动生成代码，比如可以自动更新 c/c++ 源代码里的结构，这样在表结构更新时可以帮助你迅速完成底层数据结构的重建。

1.2 安装

```
pip install db2text
```

执行代码为 dbt，可以增加一个参数文件名作为执行文件，无参数则读取当前目录下的 dbt.txt 文件。

2 dbt.txt 配置文件格式

所有 # 号开头的行为注释行，忽略其内容。

以: 开头的行为段开始，后续直到另一个段开始的内容均属于这个段。文件开头为 global 段，不需要定义段开始。

各段的内容根据不同的段会有差异，扫描执行文件时会把每一行符合 xxx=yyy 格式的内容解析出来做为参数在调用段函数时使用。一段内容全部扫描结束后开始执行这一段。内容有重复定义以后面的为准。

不同的数据库各段可能会有差异，不过不同的驱动器会尽可能保持一致，以方便用户切换数据库时学习成本较低。

执行文件中段的数量没有限制，执行时依次处理各段。

2.1 global 段

此段定义一些全局变量，目前支持的全局变量有：

- coding=gbk

指定当前的配置文件的字符集，默认为 utf8

2.2 connnect 段

首先要有一行

- driver=mysql

指定连接到哪个数据库，根据数据库的不同，后续的参数会有差异。
考虑增加使用 dbcfg 方法，这里的配置就容易一些，只用一行比如

- dbcfg=mysql234

就可以了，后续会自动使用 dbcfg 来进行数据库连接。

2.2.1 mysql 配置示例

2.3 readdata 段

读入数据库结构，具体数据格式细节随驱动器不同会有差异。
table= +/- 用空格隔开的表名，支持 python 正则表达式，如.* 匹配所有表
默认情况下处理所有的对象，如果想处理部分对象，可以在这里指定，+ 后面跟的对象是需要处理的，-后面跟的对象是不需要处理的。+/-可以多次使用，按照设置的顺序依次检查。不设置等同于 + .*
owner= 属主用于 oracle 等数据库需要读取的对象属主和连接数据库的用户不同时使用。
读入数据保存在 dbdata 字典中。
详情参考下面《设计思路/读数据库模块设计/readdata 方法》章节内容。

2.4 export 段

将读入的数据库结构导出成文本文件。
这些文件按类组织，如表放在 TABLE 目录下，视图放在 VIEW 目录下。目录之下一个对象独立为一个文件。这些文件保存为.sql，尽可能保证是相应对象在数据库中建立的格式。

2.5 render 段

file= 文件名行告诉执行器处理哪一个文件。
help=y 会让执行器输出传递给 jinja2 渲染的数据内容，方便写模板时参考。
start= 和 end= 这两行告诉执行器在文件中搜索需要替换的文件内容的头和尾。头和尾可以是同一行，这样可以适应单行方式。搜索以 python 的 re 正则模块按行搜索。从文件头开始，搜索到 start 行之后继续搜索 end，start 和 end 以及中间的所有行就是找到的需要替换的文本块。
start 和 end 行中间的内容约定是 jinj2 模板，使用 dbdata["c"]["TABLE"] 下对应表的那个字典去渲染。渲染后得到的结果判定是否需要和首尾行合并，判断的方法是取首、尾行分别和 start、end 行进行正则匹配，如果不匹配，则加上相应的首尾行。
这个设计的应用场景类似我要定义一个结构，首行是
struct stru_ 表名 { //表的说明信息
这里的表名和说明信息都是变化的，如果修改了表的说明信息，那么下一次搜索替换的时候就找不到老的结构定义位置了，所以查找首行的时候 start 行不能写表的说明信息只能写类似
struct stru_{{ tname }} {
这样，然后在模板的第一行写上类似
struct stru_{{ tname }} { //{{ tdesc }}
这样搜索的时候可以正确找到描述不同的首行，替换的时候也可以保证首行被正确修改。
最终得到的结果和要替换的文本块进行比较，如果有差异，则重写文件，用新内容替换掉老的内容。
因为要处理多张表，所以 start 和 end 会先用表名 {{ name }} 和表注释 {{ desc }} 渲染一下。
如果 start 行搜索不到，则在文件的结尾追加内容

2.6 end 段

发现这个段就意味着执行器停止扫描，这样可以把一些暂时不用的代码放在 end 段后面。
这个段最好写上，因为执行器的"特性"，实际上它并不处理最后一个段。。。。

3 设计思路

dbt 执行时根据 dbt.txt 里的内容执行相应处理。主要流程就是连接数据库、读取数据库结构、导出结构或者根据结构调整文本。这里连接数据库、读取数据库结构考虑设计成模块化的，这样可以方便替换不同的数据库。后续导出结构或者根据结构调整文本其实不同的数据库是基本类似的，差异可能只是数据结构有不同，所以这一部分设计成通用的。

3.1 读数据库模块设计

系统已经有一些模块可以用来处理 oracle、mysql、mssql，如果有其它数据库要处理，可以在 https://gitee.com/chenc224/dbt/issues 提需求，也可以根据模块的设计思路自己写一个。

模块使用 python 编写，定义在 dababase2text 目录下。命名方法建议使用数据库名称 + 版本这样的方式如 oracle11.py。

考虑在最前面定义

```
import database2text.tool as dbtt
from database2text.tool import *

__all__=["connect","readdata"]
```

模块至少提供 connect 和 readdata 方法供调用。

3.1.1 connect 方法

入口参数是一个字典，列出 connect 段中数据，比如 connect 段如下

```
:connect
driver=mssql
dbinfo={"server":"库地址", "database":"库名", "user":"用户", "password":"密码", "port":1433, "readonly":tr
dbcfg=dbname
```

则入口参数字典数据是：

{'driver': 'mssql', 'dbinfo': '{"server":"库地址","database":"库名","user":"用户","password":"密码","port":1433, "readonly":true}', 'dbcfg': 'dbname'}

注意上面这个只是用来举例，实际上 dbinfo 和 dbcfg 用一个就可以了。

3.1.2 readdata 方法

入口参数是一个字典，列出 readdata 段中数据，里面会有一些参数控制读取的内容

返回数据放在多级字典中，如果是 dbt 调用，会把它放入 dbtt.dbdata。千万注意不同的数据库，不同的驱动器版本，返回的结果有可能有比较大的差异，以下说明仅仅是一种粗略的约定。

- dbdata["sql"] 保存各对象的生成脚本，比如 dbdata["sql"]["TABLE"]["表名"] 是这张表的创建脚本。export 使用这个数据
- dbdata["exp"] 保存了 render 功能需要的一些数据，如 dbdata["exp"]["TABLE"] 是一个列表，包含所有表的数据，这个列表每一行是一个字典，保存一张表的数据：

{"tname":"表名","tdesc":"表的描述信息","ori": 从数据库中获取的原始信息，具体细节跟数据库相关, "c": 用于 c/c++ 的字段信息列表,"md": 用于输出 markdown 格式的字段数据}

ori,c,md 都是列表，每个字段一条数据字典。

c 字典包含以下内容：

{"cname": 字段名,"ns": 字段名以及长度，主要用于 char 类型定义长度用,"cdesc": 描述信息,"type": 数据类型，如 double,char,int 等}

md 字典包含以下内容：

{"name": 字段名,"desc": 注释,"type": 数据类型, "size": 字段长度,"null":True of False，是否可为空,"default": 默认值}