CodeDump配置代码生成方案

一种通用的代码生成方案

# 概述

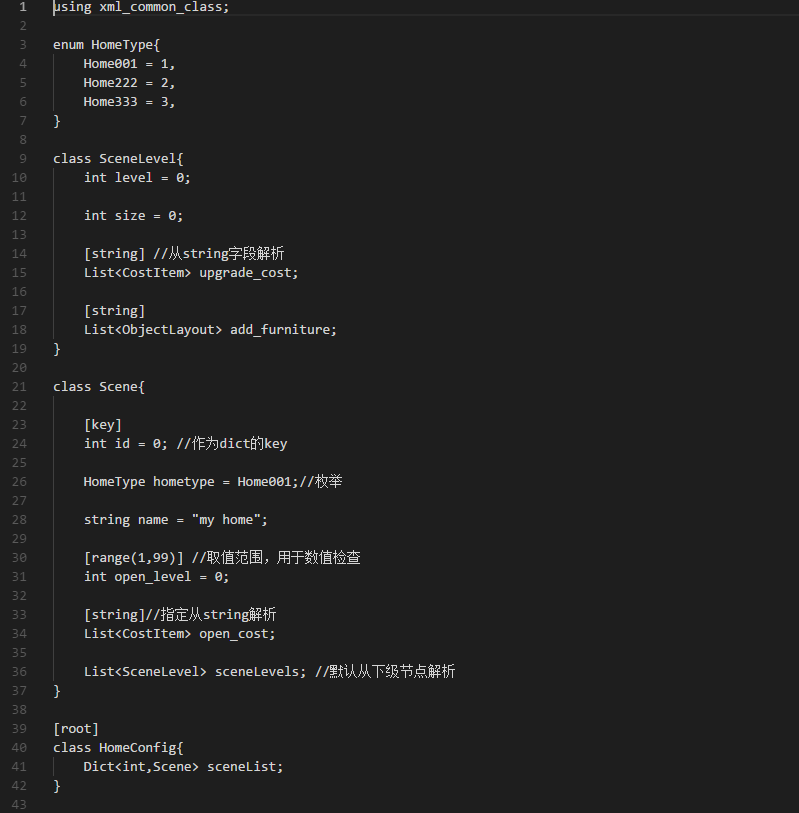
在处理配置文件生成代码的方案时，设计了一种通过数据结构描述文件（meta文件）和目标代码模板文件（template文件）生成目标代码的方案。基本思路是先解析meta文件得到数据结构信息，然后解析template文件得到目标代码的规则信息，对template文件展开，填充meta信息，就得到了目标代码。

# CodeDump工具使用规则

以xml解析生成C#代码为例介绍使用规则。这里使用home\_config.xml。



## 填写xml对应的meta文件



说明：

Home\_config.idl即为描述类型信息的meta文件。

采用类似C#语法的规则，Dictionary比较长，简写成了Dict，目前支持类（class）、枚举（enum）、基本数据类型、容器（只支持List和Dict）、特性（attribute）、注释、默认值、引用（using）。

特性是为了辅助生成代码使用的，比如[key]标记的字段是类的key字段，用于解析Dict的时候找到key的名字。

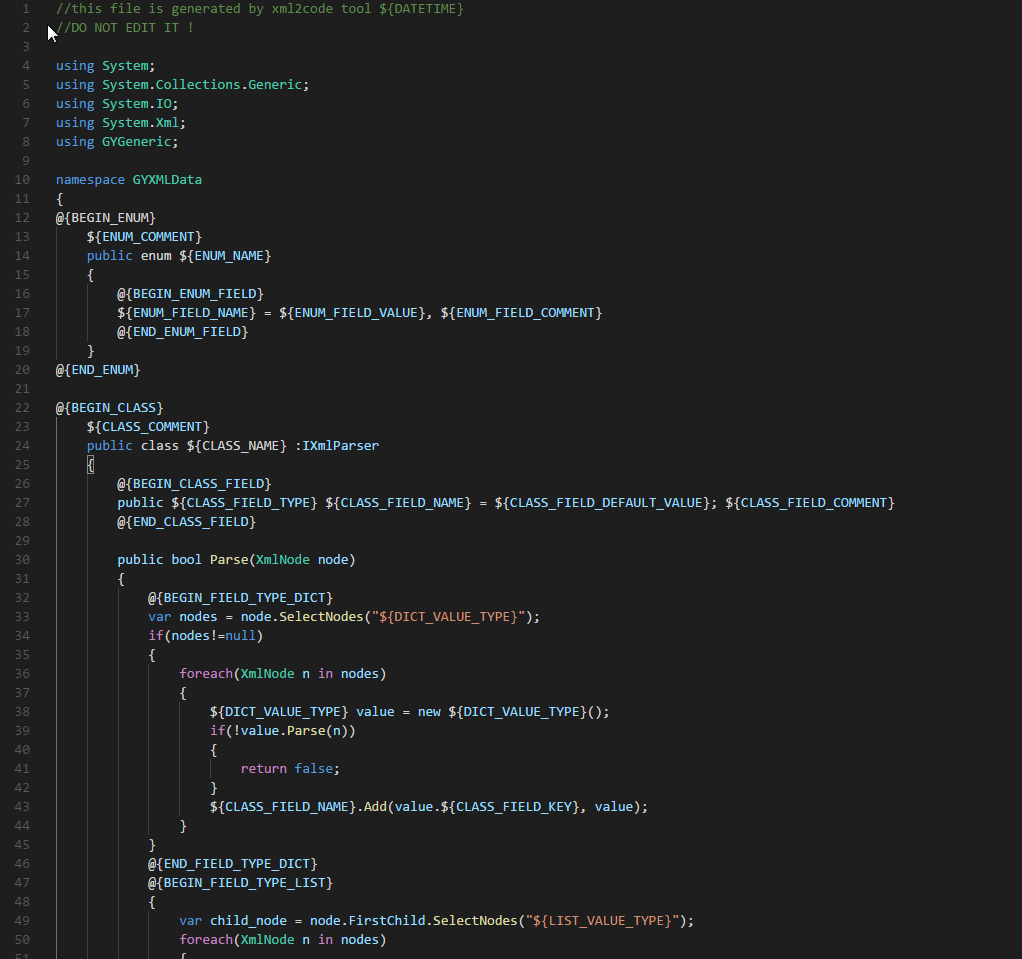
[string]表示这个字段从xml节点的一个属性字符串解析，而不是从一个xml节点解析。

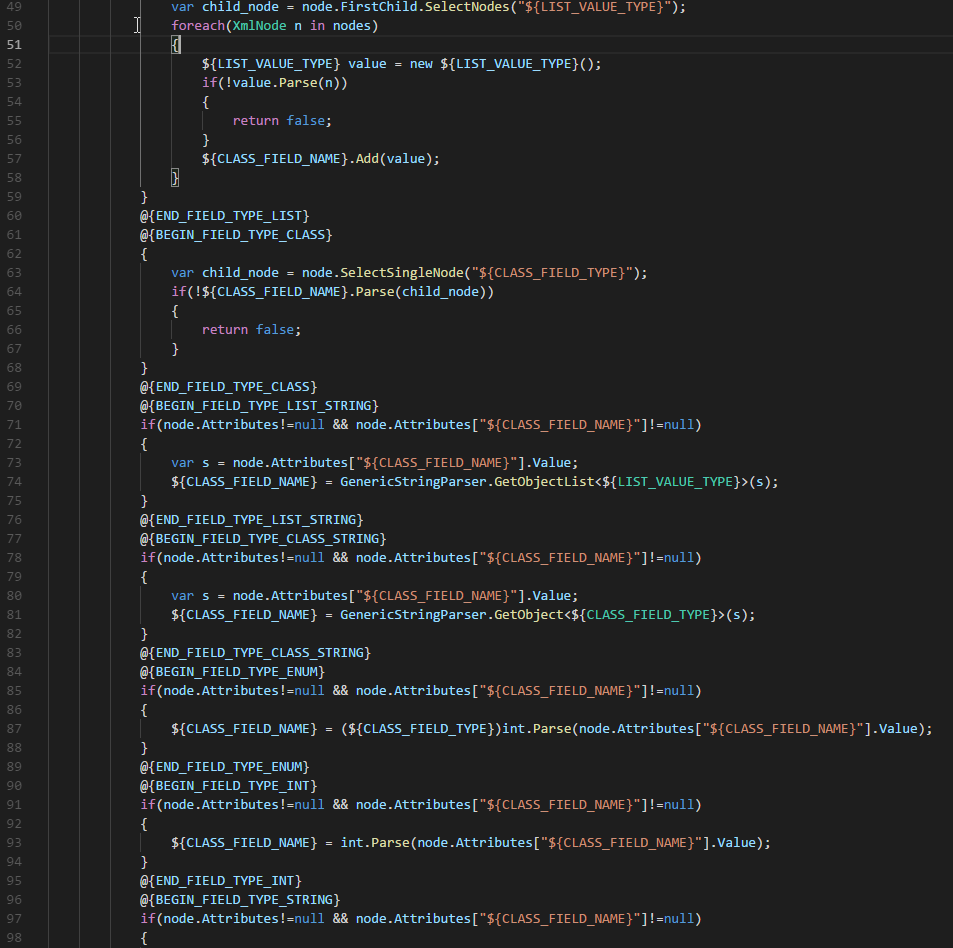
[range]表示这个字段要检查值范围，暂时没实现，不要使用。

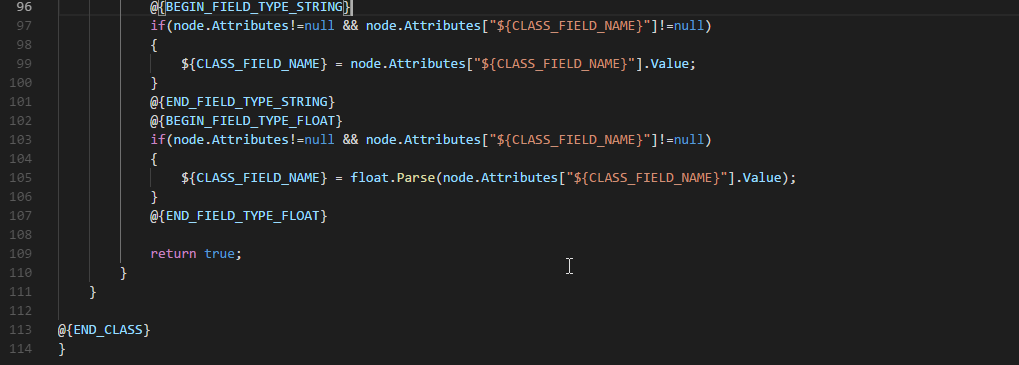
[root]表示这个类是配置文件的根类型，在生成C++代码时需要知道根类型。

Using引用的meta文件会在C++中解析为#include 头文件。

## 填写目标代码template文件







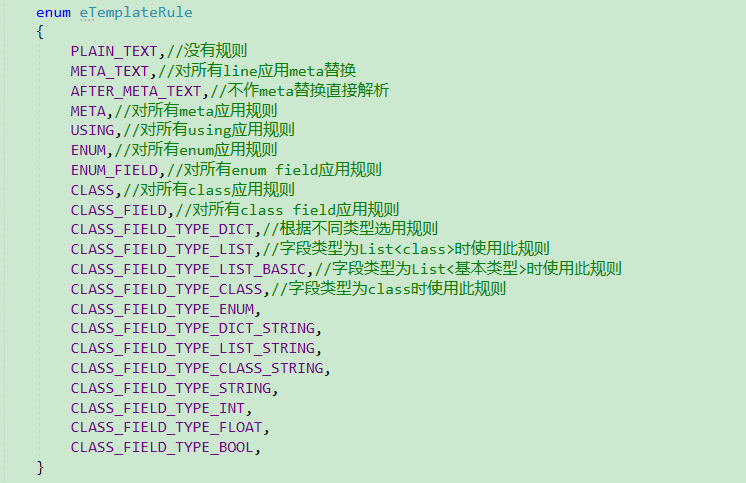
说明：

这是生成C#代码的模板文件，其基本规则如下：

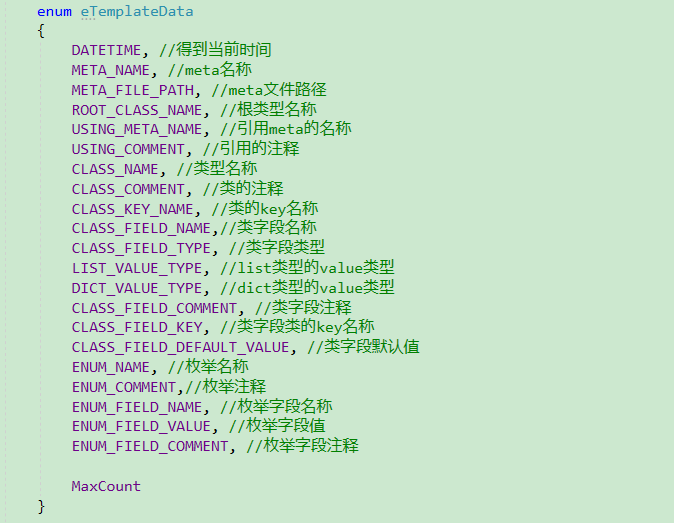
@{\w+} 为“规则标记”，通常成对出现，@{BEGIN\_XXX} @{END\_XXX}，用来指示中间的文本展开规则。规则的分类和数据结构有关，从meta级别到text级别处理方式不同。

${\w+} 为“数据标记”，数据处理和规则对应，每个级别的规则都有相应级别的数据，比如${META\_NAME}表示meta级别的名称，${CLASS\_FIELD\_NAME}为类字段级别名称。

目前支持的规则标记



目前支持的数据标记

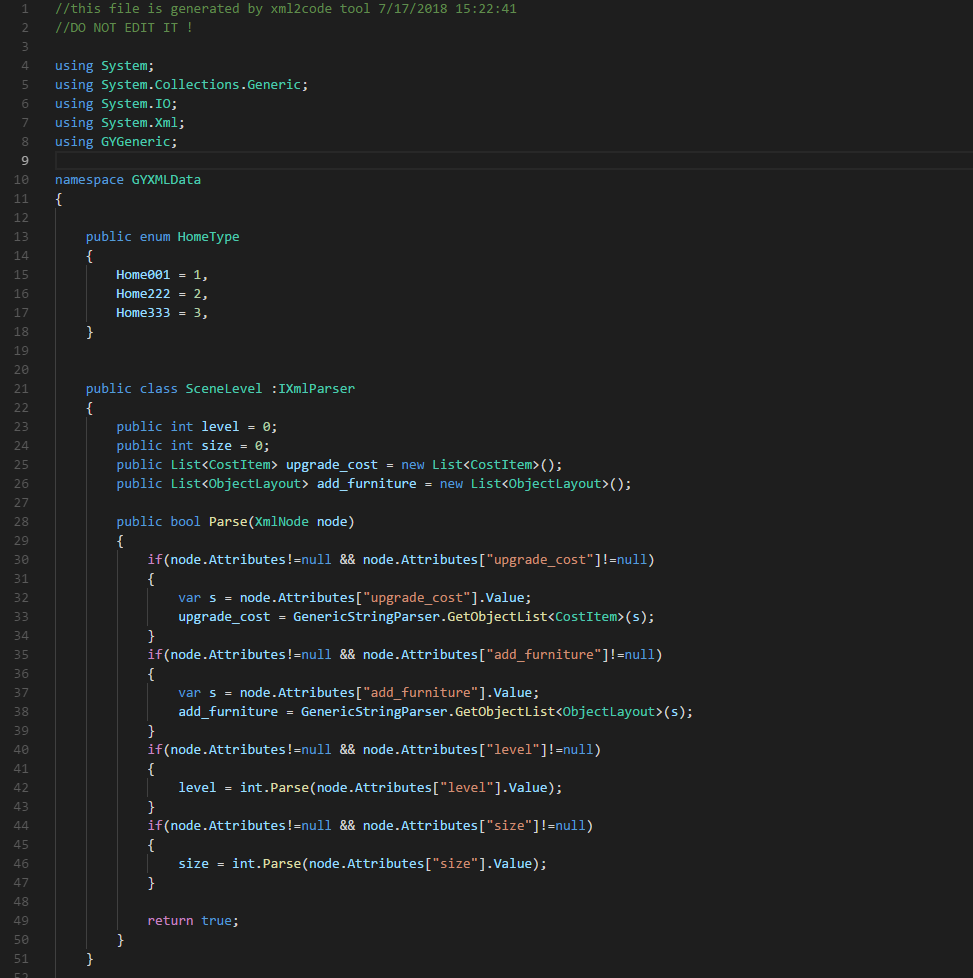


## 使用CodeDump工具生成代码

CodeDump工具使用同名json文件作为配置文件，解析并遍历配置的idl文件目录，使用模板文件生成代码到指定目录。CodeDump.json配置如下：



本例中生成的C#代码为home\_config.cs内容如下（部分）



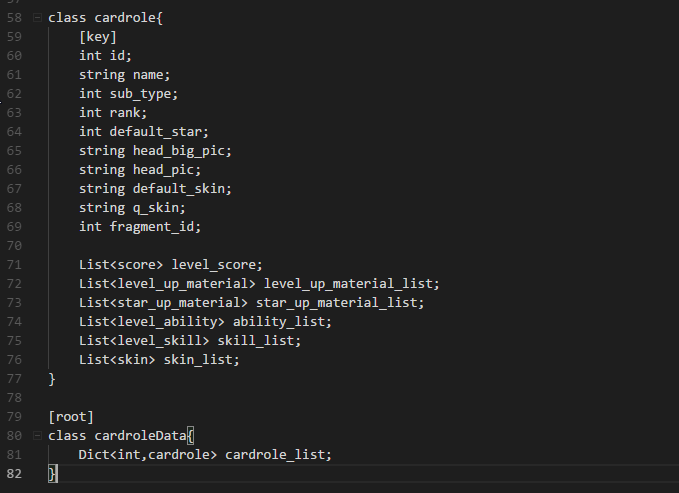
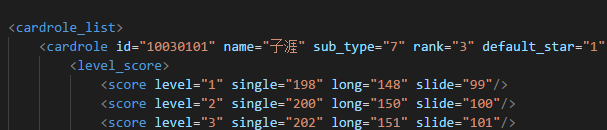
# IDL填写规范

1. 大括号不能换行。
2. 目前支持类（class）、枚举（enum）、基本数据类型（int,bool,string,float）、容器（只支持List和Dict）、特性（attribute）、注释、默认值、引用（using）。
3. 需要指定xml的root节点对应的类为[root]。
4. 需要指定dict<k,v>的v类中作为key的字段为[key]。
5. 不存在Dict<int,int>这种配置。存在List<int>配置。
6. [string]表示这个字段（class/list）从”1,2,3;4,5,6”这种字符串解析。不指定的话默认从下级节点解析。字符串分隔符是comma(,)和simicolon(;)
7. [string]还可以修饰类，默认类从xml节点解析，添加此特性则允许其从string解析。需要配合以[string]修饰的字段使用。
8. 类型有依赖关系，所以被依赖的class要写在前面，如果引用了其他idl文件里的类型需要在文件开始的地方引用一下比如：using common\_meta。

# Xml填写规范

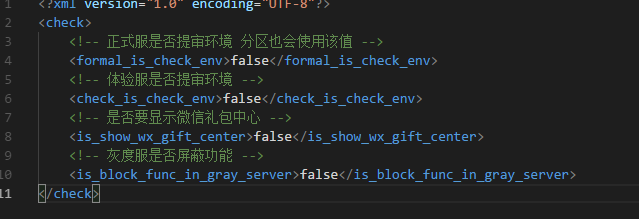
1. 尽量不要混搭风格！节点尽量写完整！
2. 每个具名节点，在IDL中填写为class、List、Dict。Class的成员对应xml中的属性。

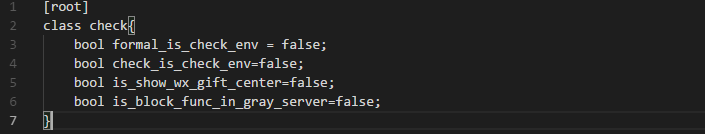
举例：



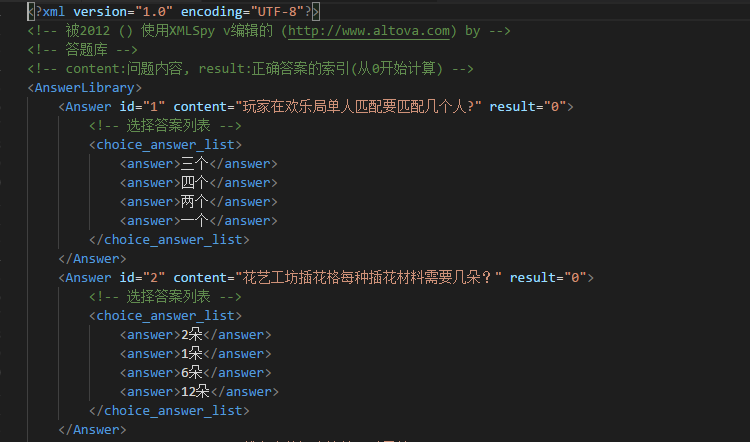
1. Class、dict、list成员如果为普通类型，也可以用inner\_text的配置方式。

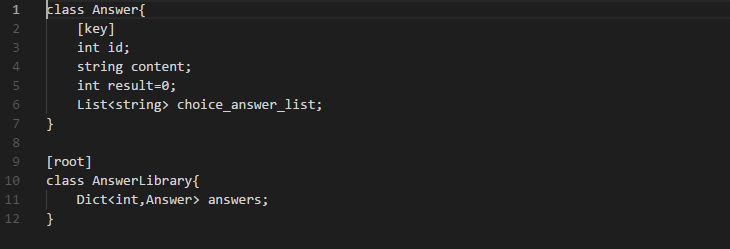
例子一：





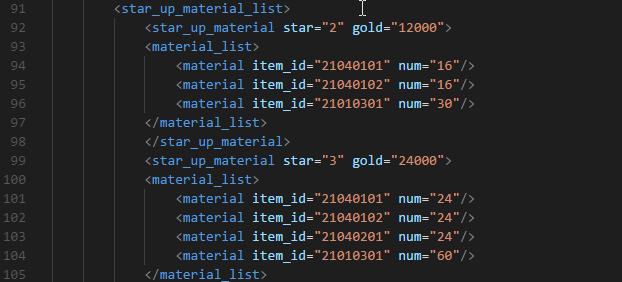
例子二：



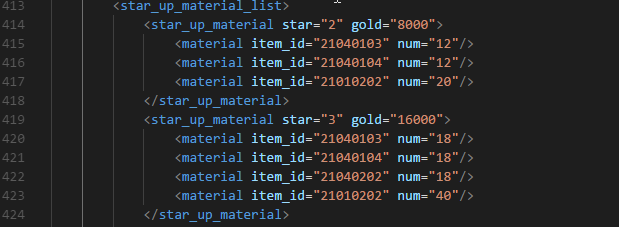


1. 可以省略dict和list的中间节点。

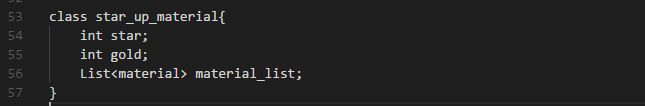
标准写法：



省略写法：

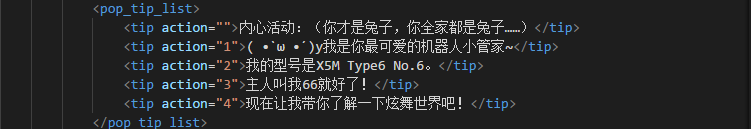


对应idl类：

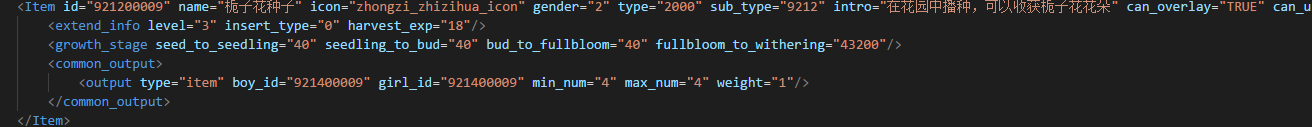


1. 不支持的配置方式：

例子一：既有attribute又有inner text，说明tip节点是个类，但丢失了一种字段描述。



例子二：不确定的配置格式，根据需求扩展节点。这种情况只能拆表，根据道具类型拆分。否则就要在idl中定义所有类型，结果会非常冗余。





例子三：不支持复杂嵌套类从string解析。混搭格式也不支持。

