转载请注明出处:<http://www.cnblogs.com/fangkm/p/3405959.html>

Chromium项目采用Grit工具来打包生成程序需要的资源,如图片资源、字符串资源等,尤其是字符串资源,牵涉到国际化的问题。Chromium为需要的资源创建单独的项目工程,工程类型为实用工具,自定义工程的生成事件, 在CustomBuild里调用grit命令,根据grd资源描述文件生成相关的资源。如chrome\_strings工程生成国际化字符串资源、chrome\_resources工程生成除字符串以外的资源,比如图片资源。

Grit工具接受grd资源描述文件,生成.h、.rc、.pak等文件。工具位于src\tools\grit文件夹下,采用python脚本编制。

src\tools\gritsettings下有个resource\_ids文件,该文件定义Chromium工程中所有的grd文件所生成的资源编号开始区段,从而确保所有的资源ID不发生冲突。该文件格式描述如下:

{

  "SRCDIR": "../..",

  "chrome/browser/browser\_resources.grd": {

    "includes": [500],

    "structures": [750],

  },

  "chrome/app/chromium\_strings.grd": {

    "messages": [11500],

  },

}

整个文件内容就是python的dict的定义,python的eval函数能直接该文件内容以字符串的方式读取,并生成dict结构,省去繁杂的解析过程, python真是一门神奇的语言。

Grit工具使用示例：

python '..\..\..\tools\grit\grit.py' '-i' ' chromium\_strings.grd' 'build' '-f' '..\..\..\tools\gritsettings\resource\_ids'

'-o' '$(OutDir)obj\global\_intermediate\ui\ui\_strings' '-D' '\_chromium' '-E' 'CHROMIUM\_BUILD=chromium' '-D' 'toolkit\_views' '-D' 'use\_aura' '-D' 'use\_ash' '-D' 'remoting' '-D' 'enable\_extensions' '-D' 'enable\_printing' '-D' 'enable\_themes'  '-D' 'enable\_app\_list' '-D' 'enable\_settings\_app' '-D' 'enable\_google\_now' '-D' 'use\_concatenated\_impulse\_responses' '-D' 'enable\_webrtc' '-D' 'enable\_mdns'

命令行参数:  -i指定grd源文件；

                 build指定生成工具,类似的工具还有buildinfo、count等等,

                        每种工具对应于grit/tool下面的相应python文件,如build对应tool下面的build.py文件,详情请参考grit\_runner.py中的相关定义

                 build后面的命令行参数都是build.py执行的的参数

目前我见到的生成示例,无论是字符串资源还是图片等其他资源,都是用build工具生成,下面就简单走一遍build.py的执行流程。

build.py主要通过提供RcBuilder类来完成解析操作. RcBuilder类派生自interface.Tool,  Tool类是各种与build同级的tools的接口类,提供有ShortDescription和Run接口。

其Run传入的命令行参数为:

-o  指定资源生成的目录

-D  指定类似于C语言的预处理宏定义,grd描述中有条件控制生成节点的逻辑,

这些定义宏可以当命令开关

-f  指定first\_ids\_file,即前面提到的gritsettings\resource\_ids文件

-E  设置grit内部使用的环境变量

Run函数内部使用grd\_reader.py的Parse方法来解析输入的grd文件,该Parse方法将xml描述的grd文件解析成树状节点结构，方法返回值为树的根节点对象GritNode。grd\_reader.py的解析流程容后细说。

在得到Parse函数返回的根节点后,运行根节点的RunGatherers方法,该方法内部遍历所有子节点,调用节点的RunPostSubstitutionGatherer(如果有的话), 目前只发现FileNode节点有此方法,用于解析对应的file元素指定的xtb文件。为了不影响整个框架流程的表述,解析xtb文件的流程也容稍后再表。

在节点树数据准备完毕后,接着就需要生成资源文件了.在RcBuilder的Process方法中遍历outputs节点下的所有output项,根据type属性指定的资源类型来选择grit.format下对应的打包工具。

如:  type="data\_package" 选择grit.format.data\_pack工具;

type="rc\_header" 选择grit.format.rc\_header工具;

其他格式参见tool/build.py代码中的\_format\_modules映射

grit.format下对应的工具会将所有的资源节点拼接成格式化的字符串并返回,  Process方法将RcBuilder将返回的格式串以相应的编码格式保存到output节点指定的文件中。

资源格式化简介:以rc\_header和data\_pack为例

rc\_header.py解析:  rc\_header是所有grd文件都必须生成的资源类型,该格式为grd的资源项生成相应的

.h文件,如chromium\_strings.h, 里面保存资源的唯一ID值,供C++代码运行期使用.

 每一项的形式为:  #define  name节点值   编号值.

该脚本内部会遍历root节点下的所有资源节点,为每个节点生成一个唯一的编号,

外部可以通过GetIds来访问这个生成的id映射，映射格式为node.GetTextualIds() : id

节点的GetTextualIds返回节点的标志性文本,一般为name属性指定的文本,即.h文件中

的#define宏名部分,所以资源的标识名不能冲突。

node的id生成规则大致如下:

1.  优先采用节点自定义的GetId方法

2.  采用group节点的first\_id属性和本节点的offset属性做累加

3.  采用group节点的first\_id属性(如果没有,则根据节点的的name

属性值做md5计算)做累加计值

data\_pack.py解析: 该格式为grd资源生成.pak格式的文件。其通过遍历每个资源节点,获取到指定语言对应的

文本value(通过节点对象的GetDataPackPair函数获取) ,将节点的id和value值以一定的格式

保存到pak文件中

**grd\_reader生成节点流程**

在介绍生成节点的流程之前,首先介绍下grd中资源节点的类型。在node. mapping.py文件中有如下映射关系:

\_ELEMENT\_TO\_CLASS = {

  'identifiers'   : empty.IdentifiersNode,

  'includes'      : empty.IncludesNode,

  'messages'      : empty.MessagesNode,

  'outputs'       : empty.OutputsNode,

  'structures'    : empty.StructuresNode,

  'translations'  : empty.TranslationsNode,

  'include'       : include.IncludeNode,

  'emit'          : io.EmitNode,

  'file'          : io.FileNode,

  'output'        : io.OutputNode,

  'ex'            : message.ExNode,

  'message'       : message.MessageNode,

  'ph'            : message.PhNode,

  'else'          : misc.ElseNode,

  'grit'          : misc.GritNode,

  'identifier'    : misc.IdentifierNode,

  'if'            : misc.IfNode,

  'part'          : misc.PartNode,

  'release'       : misc.ReleaseNode,

  'then'          : misc.ThenNode,

  'structure'     : structure.StructureNode,

  'skeleton'      : variant.SkeletonNode,

}

左边的key是grd文件中xml节点名,右边是grit生成的python对象,外部通过ElementToClass方法来访问该映射,从而决定生成什么内存对象。节点的继承结构简略如下:这里仅仅列举等下需要讲解的节点以及节点的方法。

Node为所有节点的基类, GritNode为树的根节点,程序可以通过此入口遍历其所有的子节点。

OutputNode节点对应grd文件中的output节点,表示要生成的资源文件

FileNode节点对应grd文件中的file节点,通常做为translations子节点,指定某种语言本地化需要的xtb翻译文件

MessageNode节点表示需要打包的文本资源项,同样的资源项还有IncludeNode和StructuresNode节点,用于保存图片等其他资源类型,由于精力有限,这里就不讨论 IncludeNode和StructuresNode节点了。

grd\_reader使用xml.sax库来解析grd文件,从xml.sax.handler.ContentHandler类派生出一个GrdContentHandler类来接收xml的解析回调. GrdContentHandler在startElement方法中根据读取到的节点名来生成与其对应的内存节点对象,之后调用节点的StartParsing做解析的相关定制处理,再调用节点的HandleAttribute方法读取节点的所有属性值。由于不熟悉python的使用, 我当时阅读此段代码时,很疑惑用栈的方式定位节点之间的父子关系,想了很一会才理解xml.sax.handler.ContentHandler类在解析某个节点开始时调用startElement方法,此时将自己压栈,结束时调用endElement方法,此时将自己出栈。在解析子节点时,父节点还没有解析完成,故父节点对应的endElement还未被调用,所以栈顶就是当前节点的父节点。

解析完毕后, grd\_reader调用GritNode根节点的AssignFirstIds方法, 解析之前指定的gritsettings\resource\_ids, 遍历所有的GroupingNode子节点(如MessagesNode、IncludesNode等节点),添加相应的first\_id属性值(由当前的grd文件名和节点的name属性值在resource\_ids中定位)。

**字符串本地化的翻译流程:**

基类Node节点有一uberclique成员,如果该成员没设定,则向上取父节点的uberclique值,直到根节点(如果没有就创建一个clique.UberClique对象)。当前只有根节点的此成员有值。

在 grd\_reader.py解析grd生成节点的时候,资源节点(如message节点)会调用tclib.py为节点的文本部分生成一个tclib.Message对象,该对象为节点的文本value生成一个唯一的标识ID,  即xtb文件中的translation节点的id部分。 如果节点指定use\_name\_for\_id属性为true,则直接用name属性值做translation的id部分, 这样就可以满足在英语中相同的单词在其他语种中需要不同的表达的场景,因为根据相同的文本生成的标识ID肯定是相同的。

文本生成标识ID的规则的代码在grit.extern.tclib.GenerateMessageId函数中,根据node的value文本和meaning属性(如果有的话)调用FP.FingerPrint做md5计算,取计算后的前16位.

根据生成的tclib.Message对象,调用UberClique的MakeClique方法生成一个MessageClique对象,并添加在UberClique内部维护的映射中。外部需要翻译message节点中对应语言的文本时,调用MessageNode的Translate方法,该方法调用MakeClique生成的MessageClique对象的MessageForLanguage方法,传入需要翻译的语言类型,返回MessageClique内部lang对应的 tclib.Translation对象。

下面探讨下MessageClique对象内部lang与tclib.Translation对象的映射是什么时候填充的。

前面提到的RcBuilder的Run方法在节点树结构生成完成后, 调用GritNode的RunGatherers方法,该方法遍历所有FileNode子节点,调用RunPostSubstitutionGatherer, 内部调用xtb\_reader.Parse函数,传入根节点uberclique成员的GenerateXtbParserCallback函数生成的回调来保存解析到的id:value,下面看看UberClique类的GenerateXtbParserCallback具体实现：

GenerateXtbParserCallback函数传入当前file节点的语言类型,返回一个回调函数,函数形式为Callback(id, structure)

第一个参数id为translation的id部分,

第二个参数structure参数为translation值部分,单词序列形式,

内部根据id和structure数据生成一个tclib.Translation对象,并保存到UberClique类的cliques\_成员里。

cliques\_成员是一个字典对象,存放文本串的id 到 MessageClique对象序列的映射

MessageClique对象内部存放{ lang : tclib.Translation }的映射