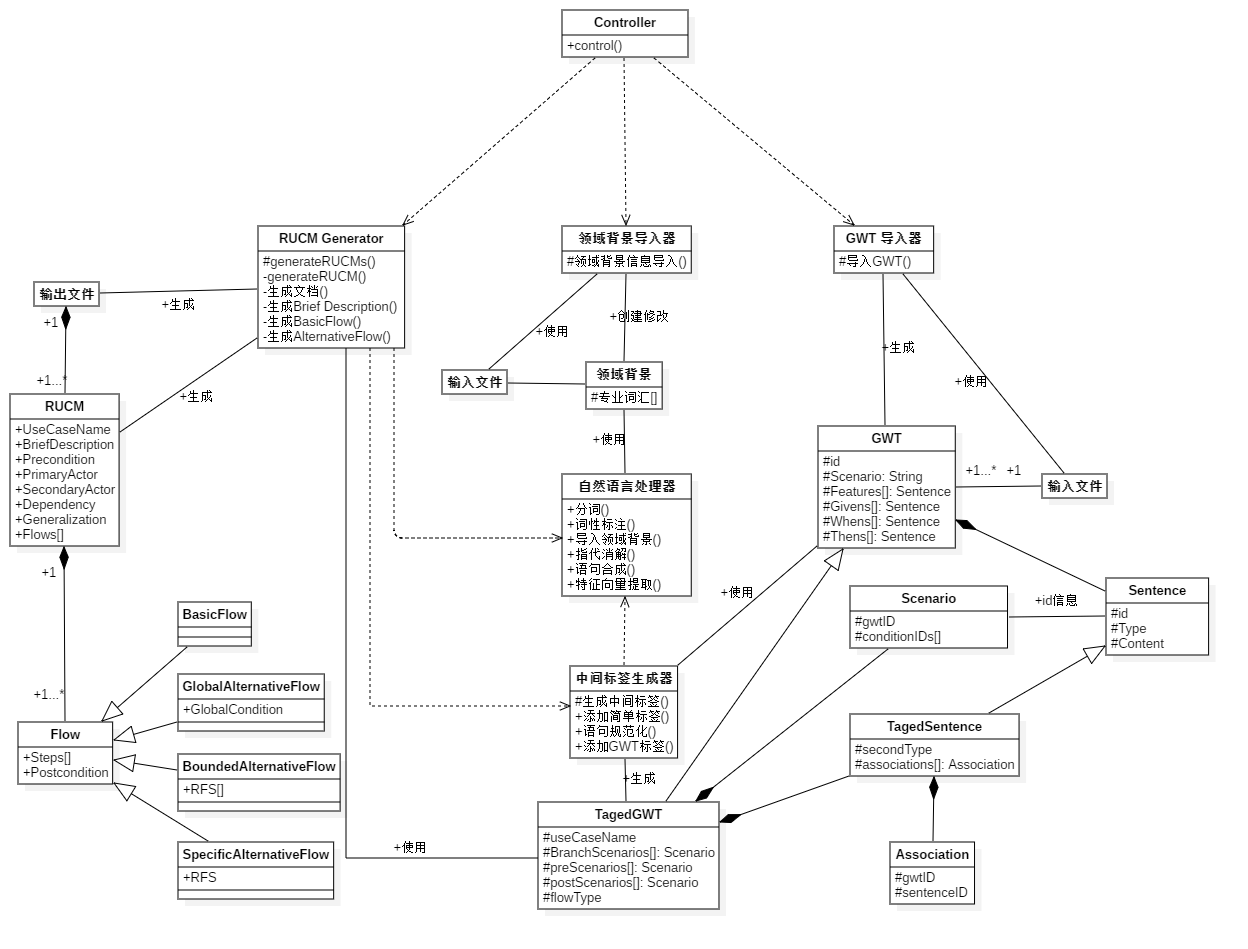
本部分将内存中数据的存储与硬盘中数据的存储联系起来，将短期存储转化为长期存储，从而提高转化效率，不需要每次转化时都将所有的GWT进行导入重新进行存储和完全的标签化。其中，RUCM是要求生成的文档，其随着GWT组合的不同而不同，会经常发生变化，所以不在数据库中进行存储。而领域背景信息可以以文件方式进行存储不需要使用数据库可以方便的进行使用。所以，目前我们仅设计了GWT的数据库存储方案。

下面是数据库生成代码与GWT相关类图：

这个图把自然语言处理器、中间标签生成器、输入文件删掉。

CREATE TABLE IF NOT EXISTS GWT\_tb(

id INT UNSIGNED AUTO\_INCREMENT,

scenario text NOT NULL,

useCaseName tinytext null,

flowType tinytext null,

PRIMARY KEY (id)

);

create table if not exists Sentence(

    id int unsigned auto\_increment,

Stype tinytext null,

secondType tinytext null,

content text not null,

sequence tinyint unsigned,

primary key (id)

);

-- engine=innodb default charset = utf8mb4

create table if not exists SentenceSentence(

    s1 int unsigned not null,

s2 int unsigned not null,

connectType tinytext not null

);

create table if not exists gwtSentence(

    gwt\_id int unsigned not null,

sentence\_id int unsigned not null,

position\_in\_gwt tinytext not null

);

我们将GWT中最小的单元视为一个个句子，每个句子都有自己的标签，同时GWT也有自己和其他GWT关系从而形成的标签。

下面，我们将按照表的视角对数据库的设计进行表述：

1. GWT\_tb
   1. Id：设计为不可重复的int类型，是GWT的独一标识方便数据库的检索
   2. Scenario：设计为text类型，用来存储字符串。理论上也需要保持独一性，但是在数据库中不进行该检查。
   3. useCaseName：加标签后，其归属的Use Case应已经规定下来，这里便是存储RUCM用例名称的位置
   4. flowType：表明该GWT在RUCM中所在的位置。
   5. 优化方向：一个GWT描述可能可以用在多个方向，从而节省存储，所以这里的useCaseName与flowType都可以放在另外的表中，与独一的GWT建立链接。
2. Sentence
   1. Id：Sentence的独一标识
   2. Stype：input、action等
   3. secondType：其他的一些标签，当前暂未定义，是为未来准备的冗余
   4. sequence：action在GWT中的顺序
3. SentenceSentence：表述句子与句子关系的表
   1. S1：句子1的id
   2. S2：句子2的id
   3. connectType：句子关系的类型，如“相同”、“相反”等
4. gwtSentence：描述GWT由哪些句子组成
   1. gwt\_id：
   2. sentence\_id:
   3. position\_in\_gwt: Given、When、Then等
   4. 优化方向：Stype、secondType、sesequence应从Sentence移动到gwtSentence中，从而将GWT和Sentence完全解耦合从而将内容相同的sentence删除，从而减少了相同句子的存储