语义模型的核心是脚本引擎，语义模型执行时都是先转换为脚本，再翻译为sql，再提交数据库执行。这里所说的“脚本”，是基于sql的扩展，可以简单理解为为：

脚本=sql+扩展

基于sql，我们提供了一系列扩展机制，包括函数扩展、规则扩展等。正是基于这些扩展，我们就可以实现封装业务复杂性，能够用sql来调用执行复杂业务逻辑，从而扩大了sql的功能。

脚本的处理机制如下图所示：



* 语法解析器

我们基于JavaCC生成了自己的解析器。利用此解析器，我们就能对表达式进行词法分析、语法分析，进而把表达式各元素对象化，以利于我们进行下面的操作。

* 规则处理器

基于sql的扩展是放在此步进行处理。规则处理器处理自定义函数、自定义表达式，最终形成面向具体数据库的sql。



规则支持扩展，自定义规则可注册在配置文件中。规则处理器会读取该文件从而加载所有配置的规则，在执行时，规则处理器会依次调用各规则。

函数规则处理是其中重要的一块，函数的扩展执行都是基于此规则来处理。

我们同样提供了函数的扩展、注册机制，函数处理器会加载该配置文件，读取所有定义的函数扩展。在执行脚本解析时，遍历访问脚本的各元素，如果判定为扩展函数，则会调用该函数的执行器进行处理。

开发人员实现自定义函数只需要做如下步骤：实现我们的函数接口，并且在配置文件中注册。然后，就可以在sql中直接使用该函数。举例来说，获取字符串长度的函数，在各数据库中函数名是不同的。SqlServer中是len()，DB2、Oracle中是length()。针对这类数据库差异，我们提供自己的函数length()实现，在sql语句中直接使用length()，规则处理器在处理Sql对象时，会遍历访问每个sql元素，如果访问到length()函数，就会调用我们的函数解析方法进行处理。在此函数的解析中，就会把该函数转换成当前数据库类型的函数。

* Sql优化器

针对sql语句进行优化，例如：消除冗余的子查询嵌套、去除无效括号、去除无效关联等。



* Sql转换器

把最终的sql对象翻译为字符串形式的sql语句。

遍历访问脚本模型各元素，返回各元素的字符串形式，拼接成完整sql