林氏物语.大器晚成之状态机代码生成框架

版权声明：

本文(或作品)由林氏原创，遵循GPL许可，你可以自由地对本文(或作品)进行任何目的的修改、转载、引用和发布，但作者不对本文(或作品)的使用负任何责任。基于此文所作的任何修改、转载、引用和发布品也需要遵循GPL许可，并要求在开头保留本声明。

联系作者：如果您有什么样的批评与指教，欢迎发email到 workspace.public@gmail.com;

如需获取最新的更新，请在email的subject填写get 010402002发送至 workspace.public@gmail.com;

如需获取林氏物语系列文章的列表，请在email的subject填写get list发送至:workspace.public@gmail.com;

如需获取林氏物语系列文章，请在email的subject填写get all发送至:workspace.public@gmail.com;

由于没有自己的邮件服务器，没有找到带附件自动回复的免费邮件服务器，采用客户端带附件自动回复，所以是非即时性回复，最迟回复时间尽量不超过一周，请勿发送多次。

l 本文编号:010402002

l 所属文集:林氏物语

l 所属类别:技术乱弹

l 所属细类:编译原理

目录

[第一章 前言 2](#_Toc359974537)

[第二章 安装篇: 5](#_Toc359974538)

[第三章 使用篇： 5](#_Toc359974539)

[第四章 乱弹篇 7](#_Toc359974540)

[第五章 引用 8](#_Toc359974541)

老子曰：“…大方无隅，大器晚成，大音希声，大象无形，道隐无名。”

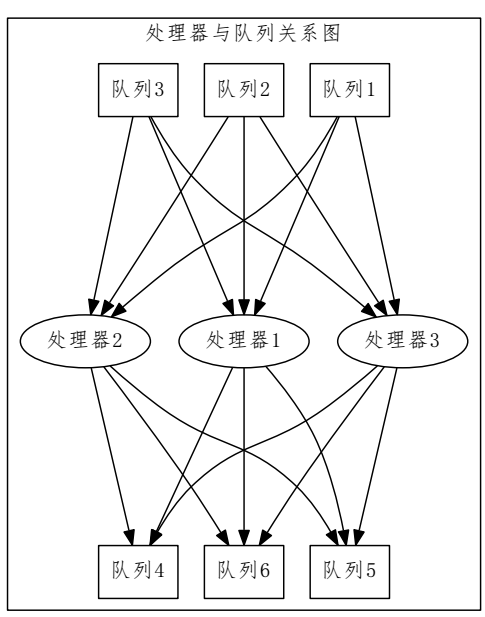
夫大器晚成者，非晚也。器之形神，已成于始，未现也，晚成者，晚现也。

--林氏按语

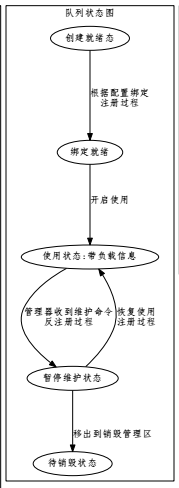
# 前言

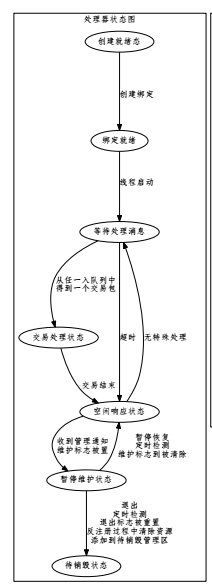
讨论组地址: <http://groups.google.com/group/antlr_chinese/>

相关的技术Q群: 281827854

这款小工具的产生来源于我的另一个产品的需求，在设计一个类似多流水线高并发的交易引擎的过程中，处理单元和数据管道队列关系如图：

队列的状态和一般处理器的状态图大致如图





希望能够将业务处理单元的开发开放给二次开发人员。但随之带来的问题是，如何能够保证二次开发人员开发的的业务处理单元的正确性和健壮性。虽然异常捕捉处理是平台必备的功能，但是尽可能将错误控制在处理器中而不让它波及环境才是治本的方法。

当描述了一些业务处理单元(业务处理器)的状态变迁过程后，我意识到了这个正确性和健壮性可以靠状态机来完成，但我觉得手工去写这类状态机是一件愚蠢的事，于是寻找一些定制状态机并生成代码的工具，在这个过程中我找到了fsme( http://fsme.sourceforge.net/ ) 这个软件，但在centos 6.2下面，我没有编译成功 :( 迫于时间关系，我没有再继续，而且fsme不能生成java代码，而那才是我所需要的，虽然没有使用过fmse 但是我通过它的文档了解了它的功能,并在代码生成时，参考了它的生成方式，在这里向fmse的作者致敬。

在比较了几款开源的状态机代码生成工具后，我觉得它们都不是我想要的。于是花了三四天写了一个自动生成的框架，定义了一套描述语法，进行了一些语义检查和代码生成，

后来在测试的时候，觉得如果有一个能够检查所有的状态迁移情况的方法更好，通过检查所有的可能性，从而保证程序的可靠性。

于是又苦憋了两天，设计了一个寻找带环路的有向图的两点之间的所有的路径的算法，根据这个来生成测试用例的代码。

最初发布的时候的代码很丑陋，是我大约花了一周左右写的，很多地方的代码结构很凌乱， 划分也不是很合理。基本像一坨shit。但它是可行的。

任何时候，make it run then make it fly or beautiful ，是我奉行的信条。

所以这次花了三四天重新重构了一版，虽然它还是个孩子，我希望能够将它打造成最好的状态机代码生成框架。也希望这款软件能够给你的工作带来便利。

最后，如果什么建议或者什么想法，也希望不吝赐教。

# 安装篇:

1、到http://www.graphviz.org/Download.php 下载一个graphviz ，并安装。

2、确保你的环境变量 path 中有 dot.exe 所在的路径的配置。检验办法 打开命令行窗口 执行 dot –version, 如果有版本信息输出则说明配置正确。

3、运行自带的demo ，准备了两个demo，一个是door的，一个是自己随便写的。在后续版本，有时间了会准备一些经典的demo。

解开压缩包，cd 到该目录，执行命令

java -cp makefsm.jar makefsm.Run .\testdata\door.data

java -cp makefsm.jar makefsm.Run .\testdata\testfsm.data

# 使用篇：

诚如demo所见，使用上是比较简单的，只需要按照规定的语法描述状态机就好了。

以下是对状态机语法规范。（熟悉LL语法的人不妨直接看 gfile目录下的makesm.g文件）。

语法：状态机描述从句;状态从句;指定开始状态从句;指定结束状态从句;事件从句;

状态机描述从句格式: 状态机类型 FSM 状态机名称 [ DESC "状态机的描述"];。

状态机的类型 有 MOORE 和 MEALY型 。

状态从句格式: status: 状态名1 [desc "状态描述1"],[状态名2 [desc "状态描述2"]...] ;。

指定开始状态从句格式:start=状态名;。

指定结束状态从句格式:end=状态名1[,状态名2...];

事件从句格式：状态名->状态名\[事件名称 [DESC "转移事件1"]\];。

例子语法解析

MOORE FSM Door  "关于门的状态机生成 详细见 http://www.ibm.com/developerworks/cn/linux/l-fsmachine/index.html ";  
STATUS: opened DESC "打开状态", closed DESC "关闭状态" ,locked "锁定状态" ,unlocked "非锁定状态", end "自定义的结束状态";  
START=opened;  
END=end;  
  
//为了保证状态机可以退出，避免陷入死循环，增加了个自定义的结束状态，并规定，门在关闭后锁定了才允许离开。  
  
  
opened->closed[closeDoor DESC "关门"];  
closed->opened[openDoor DESC "开门"];  
closed->locked[lockDoor DESC "锁门"];  
locked->unlocked[unLockDoor DESC "不锁门"];  
unlocked->locked[lockDoor2 DESC "锁门"];  
unlocked->opened[openDoor2 DESC "解锁后开门"];  
locked->end[finish DESC "完结"];

# 乱弹篇

使用状态机的方式可以获得的好处:

其实,并不存在非得要用状态机的必要情况.只能说当你可以形式化的武器后，你能够确定你所做的事情是正确的。

1. 使用系统的整体的观点去思考问题。

使用状态机的模式去思考问题，会强迫设计者必须从全盘去考虑，一步步的将复杂目标系统的状态分解，而不是将整个思路散落于支离破碎的if else中。一个好的具有活力的设计是贯穿系统的整个生命周期的，会跟随着不同的阶段而演化。

一个具有活力的设计，不管初始如何粗糙，微小，都包含着完整的特征信息。

就像一棵种子长大到一个参天大树，这个过程自始至终都携带着完整的生命信息，时间所做的，不过是将这些细节依次展开。

一个具有活力的，能够自我保持设计，也如此，在它的开始，就隐含有未来的一切关键特征的元信息，这种信息在初期未必是清晰可见的，

就像一朵花的形状和香味在它的种子期间就已经决定了，在种子期,你是看不到的，哪怕是用最高端的电子显微镜，但只要经过时间的催化，它们必定按照这种设定的方式演化（按语中的大器晚成即是此意）。

采用状态机的方式，能够在系统演化时提供快速响应和正确的保证。

1. 在统一的规则作用下，将更多的精力投入到状态响应行为上。

让实现人员更多的集中在行为的实现上，而不是行为发生序列的控制上。

1. 具备形式化推导的能力。

状态机本身就是一个形式化的工具，系统的状态可以根据输入序列明确的推导出来。另外还可以使用一些数学上的知识来辅助工作，比如本软件的生成全部可能的路径的测试用例，就用了图论相关的知识。

本软件可能适合的场景:

1. 交易引擎的处理单元的实现。
2. 游戏的npc的状态控制。
3. 其他的...暂时没想到。

# 引用

关于源码的使用，推荐使用带antlr插件的eclipse编译，环境搭建，请参见

(<http://code.google.com/p/antlr-demo/downloads/list>)