**Rime with Schemata**

Rime 输入方案创作者的第一本参考书

※ 佛振 <chen.sst@gmail.com> 修订于 2013年5月4日

**Rime 输入方案**

作者的一点主张：选用适合自己的输入法。

来折腾 Rime 的同学，我只能说，您对输入法的追求比较与众不同啦！

当然，因为您理想中的输入方式千奇百怪、也许从没有人那样玩过，所以不可能在那种勾勾选选的界面上做得出来；需要亲手来创作——

Rime 输入方案 ！

**Rime 是啥**

Rime 不是一种输入法。是从各种常见键盘输入法中提炼出来的抽象的输入算法框架。因为 Rime 涵盖了大多数输入法的「共性」，所以在不同的设定下，Rime 可化身为不同的输入法用来打字。

**Rime 输入方案又是啥**

要让 Rime 实现某种具体输入法的功能，就需要一些数据来描述这种输入法以何种形式工作。即，定义该输入法的「个性」。

如「汉语拼音」、「注音」、「仓颉码」、「五笔字型」，这些方法可凭借 Rime 提供的通用设施、给定不同的工作参数来实作。以本文介绍的规格写成一套套的配方，就是 Rime 输入方案。

**为啥这么繁？**

一键就搞掂，必然选项少，功能单一。不好玩。

输入法程序一写两三年，也许还不够火候；花两三个小时来读入门书，已是输入法专业速成班。

**准备开工**

Rime 是跨平台的输入法软件，Rime 输入方案可通用于以下发行版：

* 【中州韵】 ibus-rime → Linux
* 【小狼毫】 Weasel → Windows
* 【鼠须管】 Squirrel → Mac OS X

取得适合你系统的最新版 Rime 输入法，印一份《指南书》，准备开工了！

**Rime with Text Files**

文本为王。 Rime 的配置文件、输入方案定义及词典文件，均为特定格式的文本文档。 因此，一款够专业的 **文本编辑器** ，是设计 Rime 输入方案的必备工具。

Rime 中所有文本文檔，均要求以 UTF-8 编码，并建议使用 UNIX 换行符（LF）。

鉴于一些文本编辑器会为 UTF-8 编码的文件添加 BOM 标记，为防止误将该字符混入文中， 莫要从文件的第一行开始正文，而请在该行行首以 # 记号起一行注释，如：

# Rime default settings

# Rime schema: My First Cool Schema

# Rime dictionary: Lingua Latina

也可继续以注释行写下方案简介、码表来源、制作者、修订记录等信息，再切入正文。

**必知必会**

Rime 输入法中，多用扩展名为「.yaml」的文本文档，这是以一种可读性高的数据描述语言—— YAML 写成。

请访问 <http://yaml.org/> 了解 YAML 文檔格式。 下文只对部分语法作简要说明，而将重点放在对语义的解读上面。

Rime 输入方案亦会用到「正则表达式」实现一些高级功能。

输入方案设计者需要掌握这份文档所描述的 Perl 正则表达式语法：

<http://www.boost.org/doc/libs/1_49_0/libs/regex/doc/html/boost_regex/syntax/perl_syntax.html>

**Rime 中的数据文件分布及作用**

除程序文件以外，Rime 还包括多种数据文件。 这些数据文件存在于以下位置：

共享文件夹

* 【中州韵】 /usr/share/rime-data/
* 【小狼毫】 "安装目录\data"
* 【鼠须管】 "/Library/Input Methods/Squirrel.app/Contents/SharedSupport/"

用户文件夹

* 【中州韵】 ~/.config/ibus/rime/ （0.9.1 以下版本为 ~/.ibus/rime/）
* 【小狼毫】 "%APPDATA%\Rime"
* 【鼠须管】 ~/Library/Rime/

共享文件夹包含预设输入方案的源文件。 这些文件属于 Rime 所发行软件的一部份，在访问权限控制较严格的系统上对用户是只读的，因此谢绝软件版本更新以外的任何修改—— 一旦用户修改这里的文件，很可能影响后续的软件升级或在升级时丢失数据。

在「部署 Rime」操作时，将用到这里的输入方案源文件、并结合用户定制的内容来编译预设输入方案。

用户文件夹则包含为用户准备的内容，如：

* 〔全局设定〕 default.yaml
* 〔发行版设定〕 weasel.yaml
* 〔预设输入方案副本〕 <方案标识>.schema.yaml
* ※〔安装信息〕 installation.yaml
* ※〔用户状态信息〕 user.yaml

编译输入方案所产出的二进制文件：

* 〔Rime 棱镜〕 <方案标识>.prism.bin
* 〔Rime 固态词典〕 <词典名>.table.bin
* 〔Rime 反查词典〕 <词典名>.reverse.bin

记录用户写作习惯的文件：

* ※〔用户词典〕 <词典名>.userdb.kct
* ※〔用户词典快照〕 <词典名>.userdb.txt、<词典名>.userdb.kct.snapshot 见于同步活页夹

以及用户自己设定的：

* ※〔用户对全局设定的定制信息〕 default.custom.yaml
* ※〔用户对默认输入方案的定制信息〕 <方案标识>.custom.yaml
* ※〔用户自制输入方案〕及配套的词典源文件

注：以上标有 ※ 号的文件，包含用户数据，您在清理文件时要注意备份！

**详解输入方案**

**方案定义**

一套输入方案，通常包含「方案定义」和「词典」文件。

方案定义，命名为 <方案标识>.schema.yaml，是一份包含输入方案配置信息的 YAML 文文件。

文档中需要有这样一组方案描述：

# 以下代码片段节选自 luna\_pinyin.schema.yaml  
  
schema:  
  schema\_id: luna\_pinyin  
  name: 朙月拼音  
  version: "0.9"  
  author:  
    - 佛振 <chen.sst@gmail.com>  
  description: |  
    Rime 预设的拼音输入方案。

首先来为方案命名。schema/name 字段是显示在〔方案选单〕中的名称。

然后——重点是——要定一个在整个 Rime 输入法中唯一的「方案标识」，即 schema/schema\_id 字段的内容。 方案标识由小写字母、数字、下划线构成。 仅于输入法内部使用，且会构成方案定义文件名的一部分，因此为了兼容不同的文件系统，不要用大写字母、汉字、空格和其他符号做方案标识。 一例：这款名为【朙月拼音】的输入方案，方案标识为「luna\_pinyin」。

方案如做升级，通过版本号（schema/version）来区分相互兼容的新旧版本。

版本号——以「.」分隔的整数（或文字）构成的字符串。

如下都是版本号常见的形式：

"1"      # 最好加引号表明是字符串！  
"1.0"    # 最好加引号表明是字符串！  
"0.9.8"  
"0.9.8.custom.86427531"  # 这种形式是经过用户自定义的版本；自动生成

然而，若对方案的升级会导致原有的用户输入习惯无法在新的方案中继续使用，则需要换个新的方案标识。

比如【仓颉五代】之于【仓颉三代】、【五笔98】之于【五笔86】，其实已是互不兼容的输入法。

schema/author ——列出作者和主要贡献者，格式为文字列表：

schema:  
  author:  
    - 作者甲 <alpha@rime.org>  
    - 作者乙 <beta@rime.org>  
    - 作者丙

schema/description ——对方案作简要介绍的多行文字。

以上 schema/schema\_id、schema/version 字段用于在程序中识别输入方案， 而 schema/name、schema/author、schema/description 则主要是展示给用户的信息。

除方案描述外，方案定义文件中还包含各种功能设定，控制着输入法引擎的工作方式。

**输入法引擎与功能组件**

论 Rime 输入法的工作流程：

按键消息→后台「服务」→分配给对应的「会话」→由「方案选单」或「输入引擎」处理……

这里要点是，有会话的概念：多窗口、多线操作嘛，你懂得的，同时与好几位 MM 聊天时，有无有好几组会话。 每一组会话中都有一部输入引擎完成按键序列到文字的变换。

Rime 可以在不同会话里使用不同输入方案。因为有「方案选单」。 方案选单本身可响应一些按键。但由于他不会写字的缘故，更多时候要把按键递给同一会话中的「输入引擎」继续处理。 方案选单的贡献，就是给用户一个便捷的方案切换接口，再把用户挑中的输入方案加载到输入引擎。

再论输入引擎的工作流程：

加载输入方案、预备功能组件；各就位之后就进入处理按键消息、处理按键消息……的循环。

响应各种按键、产生各类结果的工作，由不同的功能组件分担。

好，看代码：

# luna\_pinyin.schema.yaml  
# ...  
  
engine:                    # 输入引擎设定，即挂接组件的「处方」  
  processors:              # 一、这批组件处理各类按键消息  
    - ascii\_composer       # ※ 处理西文模式及中西文切换  
    - recognizer           # ※ 与 matcher 搭配，处理符合特定规则的输入码，如网址、反查等  
    - key\_binder           # ※ 在特定条件下将按键绑定到其他按键，如重定义逗号、句号为候选翻页键  
    - speller              # ※ 拼写处理器，接受字符按键，编辑输入码  
    - punctuator           # ※ 句读处理器，将单个字符按键直接映像为文字符号  
    - selector             # ※ 选字处理器，处理数字选字键、上、下候选定位、换页键  
    - navigator            # ※ 处理输入栏内的光标移动键  
    - express\_editor       # ※ 编辑器，处理空格、回车上屏、回退键等  
  segmentors:              # 二、这批组件识别不同内容类型，将输入码分段  
    - ascii\_segmentor      # ※ 标识西文段落  
    - matcher              # ※ 标识符合特定规则的段落，如网址、反查等  
    - abc\_segmentor        # ※ 标识常规的文字段落  
    - punct\_segmentor      # ※ 标识句读段落  
    - fallback\_segmentor   # ※ 标识其他未标识段落  
  translators:             # 三、这批组件翻译特定类型的编码段为一组候选文字  
    - echo\_translator      # ※ 没有其他候选字时，回显输入码  
    - punct\_translator     # ※ 转换标点符号  
    - script\_translator    # ※ 脚本翻译器，用于拼音等基于音节表的输入方案  
    - reverse\_lookup\_translator  # ※ 反查翻译器，用另一种编码方案查码  
  filters:                 # 四、这批组件过滤翻译的结果  
    - simplifier           # ※ 繁简转换  
    - uniquifier           # ※ 过滤重复的候选字，有可能来自繁简转换

注：除示例代码中引用的组件外，尚有

- fluency\_editor    # ※ 句式编辑器，用于以空格断词、回车上屏的【注音】、【语句流】等输入方案，替换 express\_editor  
- chord\_composer    # ※ 和弦作曲家或曰并击处理器，用于【宫保拼音】等多键并击的输入方案  
- table\_translator  # ※ 码表翻译器，用于仓颉、五笔等基于码表的输入方案，替换 script\_translator

输入引擎把完成具体功能的逻辑拆分为可装卸、组合的部件。 「加载」输入方案，即按该处方挂接所需的功能组件、令这些组件从输入方案定义中加载各自的设定、准备各司其职。 而他们接下来要完成的作业，由引擎收到的一份按键消息引发。

**理解 Processors**

输入引擎，作为整体来看，以按键消息为输入，输出包括三部分：

* 一是对按键消息的处理结果：操作系统要一个结果、这按键、轮入法接是不接？
* 二是暂存于输入法、尚未完成处理的内容，会展现在输入法候选窗中。
* 三是要「上屏」的文字，并不是每按一键都有输出。通常中文字都会伴随「确认」动作而上屏，有些按键则会直接导致符号上屏，而这些还要视具体场景而定。

那么第一类功能组件 processors，就是比较笼统地、起着「处理」按键消息的作用。

按键消息依次送往列表中的 processor，由他给出对按键的处理意见：

* 或曰「收」、即由 Rime 响应该按键；
* 或曰「拒」、回禀操作系统 Rime 不做响应、请对按键做默认处理；
* 或曰这个按键我不认得、请下一个 processor 继续看。

优先级依照 processors 列表顺序排定，接收按键者会针对按键消息做处理。

虽然看起来 processor 通过组合可以承担引擎的全部任务，但为了将逻辑继续细分、Rime 又为引擎设置了另外三类功能组件。这些组件都可以访问引擎中的数据对象——输入上下文，并将各自所做处理的阶段成果存于其中。

processor 最常见的处理，便是将按键所产生的字符记入上下文中的「输入码」序列。 当「输入码」发生变更时，下一组组件 segmentors 开始一轮新的作业。

**理解 Segmentors**

Rime 可将文字、数字、符号等不同内容连续输入，此时需要识别不同格式的输入码、将输入码分成若干段分而治之。 这通过数轮代码段划分操作完成。每一轮操作中、一众 segmentors 分别给出起始于某一处、符合特定格式的代码段，识别到的最长代码段成为本轮划分的结果，而给出这一划分的一个或多个 segmentor 组件则可为该代码段打上「类型标签」；从这一新代码段的结束位置，开始下一轮划分，直到整个输入码序列划分完毕。

举例来说，【朙月拼音】中，输入码 2012nian\，划分为三个编码段：2012（贴number卷标）、nian（贴abc标签）、\（贴punct标签）。

那些标签是初步划分后判定的类型，也可能有一个编码段贴多个卷标的情况。下一个阶段中，translators 会把特定类型的编码段翻译为文字。

**理解 Translators**

顾名思义，translator 完成由编码到文字的翻译。但有几个要点：

* 一是翻译的对象是划分好的一个代码段。
* 二是某个 translator 组件往往只翻译具有特定标签的代码段。
* 三是翻译的结果可能有多条，每条结果成为一个展现给用户的候选项。
* 四是代码段可由几种 translator 分别翻译、翻译结果按一定规则合并成一列候选。
* 五是候选项所对应的编码未必是整个代码段。用拼音敲一个词组时，词组后面继续列出单字候选，即是此例。

双目如探针般进入内存查看，发现翻译的结果呈现这种形式：

input | tag    | translations  
------+--------+-------------------------------------  
2012  | number | [ "2012" ], [ "二〇一二" ]  
nian  | abc    | [ "年", "念", "念",... ], [ "nian" ]  
\     | punct  | [ "、", "\" ]

输入串划分为多个代码段、每段代码又可具有多组翻译结果；取各代码段的首选结果连接起来，就是预备上屏的文字「2012年、」。

且将以上所示的数据称为「作文」。这是一篇未定稿（未搞定）的作文，输入法接口此时显示预备上屏的文字「2012年、」，并列出最末一个代码段上的候选「、」及「\」以供选择。

有两款主力 translator 完成主要文字内容的翻译，其实现的方式很不一样：

* r10n\_translator 也叫做 script\_translator 修炼罗马字分析法，以「固定音节表」为算法的基础，识别输入码的音节构成，推敲排列组合、完成遣词造句。
* table\_translator 修炼传承自上世纪的码表功夫，基于规则的动态码表，构成编码空间内一个开放的编码集合。

拼音、注音、方言拼音，皆是以固定音节表上的拼写排列组合的方式产生编码，故适用罗马字分析法。 仓颉、五笔字型这类则是传统的码表输入法。

如果以码表方式来写拼音输入方案，是怎样的效果呢？虽然仍可完成输入，但无法完全实现支持简拼、模糊拼音、使用隔音符号的动态调频、智能语句等有用的特性。

反之，以罗马字方式使用码表输入法，则无法实现定长编码顶字上屏、按编码规则构词等功能。在 Rime 各发行版预设输入方案中，有一款「速成」输入方案，即是以 r10n\_translator 翻译仓颉码，实现全、简码混合的语句输入。

概括起来，这是两种构造新编码的方式：罗马字式输入方案以一组固定的基本音节码创造新的组合而构词，而码表式输入方案则以一定码长为限创造新的编码映像而构词。

**理解 Filters**

上一步已经收集到各个代码段的翻译结果，当输入法需要在接口呈现一页候选项时，就从最末一个代码段的结果集中挑选、直至取够方案中设定的页最大候选数。

每从结果集选出一条字词、会经过一组 filters 过滤。多个 filter 串行工作，最终产出的结果进入候选序列。

filter 可以：

* 改装正在处理的候选项，修改某些属性值：简化字、火星文、菊花文有无有？过时了！有 Rime，你对文字的想象力终于得救
* 消除当前候选项，比如检测到重复（由不同 translator 产生）的候选条目
* 插入新的候选项，比如根据已有条目插入关联的结果
* 修改已有的候选序列

**码表与词典**

词典是 translator 的参考书。

他往往与同名输入方案配套使用，如拼音的词典以拼音码查字，仓颉的词典以仓颉码查字。但也可以由若干编码属于同一系统的输入方案共享，如各种双拼方案，都使用和拼音同样的词典，于是不仅复用了码表数据，也可共享用户以任一款此系列方案录入的自造词（仍以码表中的形式即全拼编码记录）。

Rime 的词典文件，命名为 <词典名>.dict.yaml，包含一份码表及对应的规则说明。 词典文件的前半部份为一份 YAML 文檔：

# 注意这里以 --- ... 分别标记出 YAML 文文件的起始与结束位置  
# 在 ... 标记之后的部份就不会作 YAML 文档来解读  
  
---  
name: luna\_pinyin  
version: "0.9"  
sort: by\_weight  
use\_preset\_vocabulary: true  
...

解释：

* name: 词典名，内部使用，命名原则同「方案标识」；可以与配套的输入方案标识一致，也可不同；
* version: 管理词典的版本，规则同输入方案定义文件的版本号；
* sort: 词条初始排序方式，可选填 by\_weight（按词频高低排序）或 original（保持原码表中的顺序）；
* use\_preset\_vocabulary: 填 true 或 false，选择是否导入预设词汇表【八股文】。

码表，定义了输入法中编码与文字的对应关系。

码表位于词典文件中 YAML 结束标记之后的部份。 其格式为以制表符分隔的值（TSV），每行定义一条「文字－编码」的对应关系：

# 单字  
你       ni  
我       wo  
的       de      99%  
的       di      1%  
地       de      10%  
地       di      90%  
目       mu  
好       hao  
  
# 词组  
你我  
你的  
我的  
我的天  
天地      tian di  
好天  
好好地  
目的      mu di  
目的地     mu di di

※注意： **不要** 从网页复制以上代码到实作的词典文件！因为网页里制表符被转换成空格从而不符合 Rime 词典要求的格式。

码表部份，除了以上格式的编码行，还可以包含空行（不含任何字符）及注释行（行首为 # 符号）。

以制表符（Tab）分隔的第一个字段是所定义的文字，可以是单字或词组；

第二个字段是与文字对应的编码；若该编码由多个「音节」组成，音节之间以空格分开；

可选地、第三个字段是设定该字词权重的频度值（非负整数），或相对于预设权值的百分比（浮点数%）。 在拼音输入法中，往往多音字的若干种读音使用的场合不同，于是指定不同百分比来修正每个读音的使用频度。

词组如果满足以下条件，则可以省去编码字段：

* 词组中每个单字均有编码定义
* 词组中不包含多音字（例：你我），或多音字在该词组中读音的权值超过该多音字全部读音权值的5%（例：我的）

这种条件下，词组的编码可由单字编码的组合推导出来。

反之，则有必要给出词组的编码以消除自动注音的不确定性（例：天地）。

当含有多音字的词组缺少编码字段时，自动注音程序会利用权重百分比高于5%的读音进行组合、生成全部可能的注音，如：

「好好地」在编译时自动注音为「hao hao de」、「hao hao di」

**设定项速查手册**

[雪斋的文档](https://github.com/LEOYoon-Tsaw/Rime_collections/blob/master/Rime_discription.md) 全面而详细解释了输入方案及词典中各设定项的含义及用法。

**八股文**

Rime 有一份名为【八股文】的预设词汇表。

多数输入方案需要用到一些标准白话文中的通用词汇。为免重复在每份码表中包含这些词汇，减少输入方案维护成本，Rime 提供了一份默认词汇表及自动编码（注音）的设施。

创作输入方案时，如果希望完全控制词汇表的内容而不采用【八股文】中的词组，只须直接将词汇编入码表即可。

否则，在词典文件中设定 use\_preset\_vocabulary: true 将【八股文】导入该词典； 在码表中给出单字码、需要分辨多音字的词组编码、以及该输入方案特有的词汇，其他交给自动注音来做就好啦。

Rime预设输入方案正是利用这份词汇表及自动注音工具，在不牺牲效果及可维护性的前提下、使词典文件压缩到最小的行数。

【八股文】包含从 CC-CEDICT、新酷音等开源项目中整理出来的约二十三万条词汇，其用字及词频数据针对传统汉字做过调整。因此基于这份词汇表产生的输入结果，比较接近以传统汉字行文的实际用法。

为了充分利用【八股文】提供的词汇，自定义的词典应保证单字码表收录了符合 opencc 字形标准的常用字。特别注意，该标准对以下几组异体字的取舍，【八股文】将这些字（包括词组及字频）统一到左边一列的字形。

为       为  
伪       伪  
妫       妫  
沩       沩  
凶       凶  
启       启  
棱       棱  
污       污  
泄       泄  
涌       涌  
床       床  
着       着  
灶       灶  
众       众  
里       里  
踊       踊  
面       面  
羣       群  
峯       峰

请务必在码表中收录左列的单字；并建议收全右列的单字。

输入法词典往往对下列几组字不做严格区分，opencc 则倾向于细分异体字的不同用法。

吃       吃  
霉       霉  
考       考  
核       核

请尽量在码表中收全以上单字。

部署过程中，未能完成自动注音的字、词会以警告形式出现在日志文件里。如果所报告的字为生僻字、您可以忽略他；如果警告中大量出现某个常用字，那么应该注意到码表里缺失了该字的注音。

**编译输入方案**

将写好的输入方案布署到 Rime 输入法的过程，称为「编译」：

为查询效率故，输入法工作时不直接加载文本格式的词典源文件，而要在编译过程中，为输入方案生成专为高速查询设计的「.bin」文件。

编译时程序做以下几件事：

* 将用户的定制内容合并到输入方案定义中，在用户文件夹生成 .schema.yaml 文檔副本；
* 依照输入方案中指定的词典：求得音节表（不同种编码的集合）、单字表；
* 对词典中未提供编码的词组自动注音，也包括从【八股文】导入的词组；
* 建立按音节编码检索词条的索引，制作 Rime 固态词典；
* 建立按词条检索编码的索引，制作 Rime 反查词典；
* 依照音节表和方案定义中指定的拼写运算规则，制作 Rime 棱镜。

**布署 Rime**

初次安装 Rime 输入法，无有任何输入方案和用户设定。因此安装的最后一个步骤即是把发行版预设的输入方案和设定文件布署到 Rime 为该用户创建的工作目录，至此 Rime 纔成为一部可以发动的输入引擎。

此后无论是修改已有方案的设定，或是添加了新的输入方案，都需要「重新布署 Rime」成功后方可使用户的新设定作用于 Rime 输入法。

〔★〕重新布署 Rime 的方法是：

* 【小狼毫】从开始菜单选择「重新部署」；或当开启托盘图标时，在托盘图标上右键选择「重新布署」；
* 【鼠须管】在系统语言文字选单中选择「重新布署」；
* 【中州韵】点击输入法状态区（或IBus菜单）上的 ⟲ (Deploy) 按钮；
* 早于 ibus-rime 0.9.2 的版本：删除用户文件夹的 default.yaml 之后、执行 ibus-daemon -drx 重载 IBus

**定制指南**

Rime 输入方案，将 Rime 输入法的设定整理成完善的、可分发的形式。 但并非一定要创作新的输入方案，才可以改变 Rime 的行为。

当用户需要对 Rime 中的各种设定做小幅的调节，最直接、但不完全正确的做法是：编辑用户文件夹中那些 .yaml 文檔。

这一方法有弊端：

* 当 Rime 软件升级时，也会升级各种配置文件、预设输入方案。用户编辑过的文文件会被覆写为更高版本，所做调整也便丢失了。
* 即使在软件升级后再手动恢复经过编辑的文件，也会因配置文件的其他部分未得到更新而失去本次升级新增和修复的功能。

因此，对于随 Rime 发行的配置文件及预设输入方案，推荐的定制方法是：

创建一个文件名的主体部份（「.」之前）与要定制的文件相同、次级扩展名（位于「.yaml」之前）写作 .custom 的定制档，形如：

patch:  
  "一级设定项/二级设定项/三级设定项": 新的设定值  
  "另一个设定项": 新的设定值  
  "再一个设定项": 新的设定值

patch 定义了一组「补靪」，以源文件中的设定为底本，写入新的设定项、或以新的设定值取代旧有的值。

以下这些例子，另载于一篇[《定制指南》](http://code.google.com/p/rimeime/wiki/CustomizationGuide)，其中所介绍的知识和技巧，覆盖了不少本文未讨论的细节，想必对于创作新的输入方案会有启发。

* [一例、定制每页候选数](http://code.google.com/p/rimeime/wiki/CustomizationGuide#%E4%B8%80%E4%BE%8B%E3%80%81%E5%AE%9A%E8%A3%BD%E6%AF%8F%E9%A0%81%E5%80%99%E9%81%B8%E6%95%B8)
* [一例、定制标点符号](http://code.google.com/p/rimeime/wiki/CustomizationGuide#%E4%B8%80%E4%BE%8B%E3%80%81%E5%AE%9A%E8%A3%BD%E6%A8%99%E9%BB%9E%E7%AC%A6%E8%99%9F)
* [一例、定制简化字输出](http://code.google.com/p/rimeime/wiki/CustomizationGuide#%E4%B8%80%E4%BE%8B%E3%80%81%E5%AE%9A%E8%A3%BD%E7%B0%A1%E5%8C%96%E5%AD%97%E8%BC%B8%E5%87%BA)
* [一例、定制方案选单](http://code.google.com/p/rimeime/wiki/CustomizationGuide#%E4%B8%80%E4%BE%8B%E3%80%81%E5%AE%9A%E8%A3%BD%E6%96%B9%E6%A1%88%E9%81%B8%E5%96%AE)

重要！创作了新的输入方案，最后一步就是在「方案选单」里启用他。

**拼写运算**

应该算是 Rime 输入法最主要的独创技术。

概括来说就是将方案中的编码通过规则映像到一组全新的拼写形式！ 也就是说能让 Rime 输入方案在不修改码表的情况下、适应不同的输入习惯。

拼写运算能用来：

* 改革拼写法
  + 将编码映像到基于同一音系的其他拼写法，如注音、拼音、国语罗马字相互转换
  + 重定义注音键盘、双拼方案
* 实现简拼查询
* 在音节表上灵活地定义模糊音规则
* 实现音节自动纠错
* 变换回显的输入码或提示码，如将输入码显示为字根、注音符号、带声调标注的罗马字

给力吗？

[★这里](http://code.google.com/p/rimeime/wiki/SpellingAlgebra) 有介绍拼写运算的专题文章。

**综合演练**

如果你安装好了Rime却不会玩，就一步一步跟我学吧。

本系列课程每个步骤的完整代码可由此查阅：

<http://code.google.com/p/rimeime/source/browse/trunk/doc/tutorial/>

**【一】破空中出鞘**

**Hello, Rime!**

第一个例子，总是最简单的（也是最傻的）。

# Rime schema  
# encoding: utf-8  
#  
# 最简单的 Rime 输入方案  
#  
  
schema:  
  schema\_id: hello    # 注意此ID与文件名里 .schema.yaml 之前的部分相同  
  name: 大家好        # 将在〔方案选单〕中显示  
  version: "1"        # 这是文字类型而非整数或小数，如 "1.2.3"

起首几行是注释。而后只有一组必要的方案描述信息。

这一课主要练习建立格式正确的YAML文檔。

* 要点一，让你的文本编辑器以UTF-8编码保存该文件；
* 要点二，注意将 schema: 之下的三行代码以空格缩进——我的习惯是用两个空格——而 **不要** 用Tab字符来缩进。

缩进表示设定项所属的层次。在他处引用到此文档中的设定项，可分别以 schema/schema\_id, schema/name, schema/version 来指称。

我现在把写好的方案文档命名为 hello.schema.yaml，丢进用户文件夹——对，只要这一个文件就妥了；

然后，启用他。有些版本会有「方案选单设定」这个接口，在那里勾选【大家好】这个方案即可。若无有设定接口，则按照上文《定制方案选单》一节来做。

好运！我已建立了一款名为【大家好】的新方案！虽然他没有实现任何效果、按键仍会像无有输入法一样直接输出西文。

**开始改装**

为了处理字符按键、生成输入码，本例向输入引擎添加两个功能组件。

以下代码仍是ID为 hello 的新款输入方案，但增加了 schema/version 的数值。 以后每个版本，都以前一个版本为基础改写，引文略去无有改动的部分，以突出重点。

# ...   
schema:  
  schema\_id: hello  
  name: 大家好  
  version: "2"  
  
engine:  
  processors:  
    - fluency\_editor  
  segmentors:  
    - fallback\_segmentor

fluency\_editor 将字符按键记入输入上下文，fallback\_segmentor 将输入码连缀成一段。于是重新布署后，按下字符键不再直接上屏，而显示为输入码。

你会发现，该输入法只是收集了键盘上的可打印字符，并于按下空格、回车键时令输入码上屏。

现在就好似写输入法程序的过程中，将将取得一小点成果，还有很多逻辑未实现。不同的是，在Rime输入方案里写一行代码，顶 Rime 开发者所写的上百上千行。因此我可以很快地组合各种逻辑组件、搭建出心里想的输入法。

**创建候选项**

第二版的【大家好】将键盘上所有字符都记入输入码，这对整句输入有用，但是时下流行输入法只处理编码字符、其他字符直接上屏的形式。为了对编码字符做出区分，以下改用 speller + express\_editor 的组合取代 fluency\_editor：

# ...  
  
schema:  
  # ...  
  version: "3"  
  
engine:  
  processors:  
    - speller          # 把字母追加到编码串  
    - express\_editor   # 空格确认当前输入、其他字符直接上屏  
  segmentors:  
    - fallback\_segmentor

speller 默认只接受小写拉丁字母作为输入码。 试试看，输入其他字符如大写字母、数字、标点，都会直接上屏。并且如果已经输入了编码时，下一个直接上屏的字符会将输入码顶上屏。

再接着，创建一个最简单的候选项——把编码串本身作为一个选项。故而会提供这个选项的新组件名叫 echo\_translator。

# ...  
  
engine:  
  # ...  
  translators:  
    - echo\_translator  # （无有其他结果时，）创建一个与编码串一个模样的候选项

至此，【大家好】看上去与一个真正的输入法形似啦。只是还不会打出「大家好」哇？

**编写词典**

那就写一部词典，码表中设定以 hello 作为短语「大家好」的编码：

# Rime dictionary  
# encoding: utf-8  
  
---  
name: hello  
version: "1"  
sort: original  
...  
  
大家好     hello  
再见      bye  
再会      bye

※注意： **不要** 从网页复制以上代码到实作的词典文件！因为网页里制表符被转换成空格从而不符合 Rime 词典要求的格式。

同时修改方案定义：

#...  
  
schema:  
  # ...  
  version: "4"  
  
engine:  
  #...  
  segmentors:  
    - abc\_segmentor       # 标记输入码的类型  
    - fallback\_segmentor  
  translators:  
    - echo\_translator  
    - table\_translator    # 码表式转换  
  
translator:  
  dictionary: hello       # 设定 table\_translator 使用的词典名

工作流程是这样的：

* speller 将字母键加入输入码序列
* abc\_segmentor 给输入码打上标签 abc
* table\_translator 把带有 abc 签的输入码以查表的方式译为中文
* table\_translator 所查的码表在 translator/dictionary 所指定的词典里

现在可以敲 hello 而打出「大家好」。完工！

**实现选字及换页**

等一下。

记得 hello 词典里，还有个编码叫做 bye。敲 bye，Rime 给出「再见」、「再会」两个候选短语。

这时敲空格键，就会打出「再见」；那么怎样打出「再会」呢？

大家首先想到的方法，是：打完编码 hello，按 1 选「再见」，按 2 选「再会」。 可是现在按下 2 去，却是上屏「再见」和数字「2」。可见并没有完成数字键选字的处理，而是将数字同其他符号一样做了顶字上屏处理。

增加一部 selector，即可实现以数字键选字。

schema:  
  # ...  
  version: "5"  
  
engine:  
  processors:  
    - speller  
    - selector         # 选字、换页  
    - navigator        # 移动插入点  
    - express\_editor  
  # ...

selector 除了数字键，还响应前次页、上下箭头键。因此选择第二候选「再会」，既可以按数字2，又可以按箭头键「↓」将「再会」高亮、再按空格键确认。

navigator 处理左右箭头键、Home、End键，实现移动插入点的编辑功能。有两种情况需要用到他：一是发现输入码有误需要定位修改，二是缩小候选词对应的输入码的范围、精准地编辑新词组。

接下来向词典添加一组重码，以检验换页的效果：

---  
name: hello  
version: "2"  
sort: original  
...  
  
大家好     hello  
再见      bye  
再会      bye  
  
星期一     monday  
星期二     tuesday  
星期三     wednesday  
星期四     thursday  
星期五     friday  
星期六     saturday  
星期日     sunday  
  
星期一     weekday  
星期二     weekday  
星期三     weekday  
星期四     weekday  
星期五     weekday  
星期六     weekday  
星期日     weekday

默认每页候选数为5，输入 weekday，显示「星期一」至「星期五」。再敲 Page\_Down 显示第二页后选词「星期六、星期日」。

**输出中文标点**

schema:  
  # ...  
  version: "6"  
  
engine:  
  processors:  
    - speller  
    - punctuator        # 处理符号按键  
    - selector  
    - navigator  
    - express\_editor  
  segmentors:  
    - abc\_segmentor  
    - punct\_segmentor   # 划界，与前后方的其他编码区分开  
    - fallback\_segmentor  
  translators:  
    - echo\_translator  
    - punct\_translator  # 转换  
    - table\_translator  
  
# ...  
  
punctuator:             # 设定符号表，这里直接导入预设的  
  import\_preset: default

这次的修改，要注意 punctuator, punct\_segmentor, punct\_translator 相对于其他组件的位置。

punctuator/import\_preset 告诉 Rime 使用一套默认的符号表。他的值 default 可以换成其他名字如 xxx，则 Rime 会读取 xxx.yaml 里面定义的符号表。

如今再敲 hello. 就会得到「大家好。」

**用符号键换页**

早先流行用 - = 这一对符号换页，如今流行用 , 和 . 。 在第六版中「，」「。」是会顶字上屏的。现在要做些处理以达到一键两用的效果。

Rime 提供了 key\_binder 组件，他能够在一定条件下，将指定按键绑定为另一个按键。对于本例就是：

* 当展现候选菜单时，句号键（period）绑定为向后换页（Page\_Down）
* 当已有（向后）换页动作时，逗号键（comma）绑定为向前换页（Page\_Up）

逗号键向前换页的条件之所以比句号键严格，是为了「，」仍可在未进行换页的情况下顶字上屏。

经过 key\_binder 的处理，用来换页的逗号、句号键改头换面为前、后换页键，从而绕过 punctuator，最终被 selector 当作换页来处理。

最终的代码如下：

schema:  
  schema\_id: hello  
  name: 大家好  
  version: "7"  
  
engine:  
  processors:  
    - key\_binder  # 抢在其他 processor 处理之前判定是否换页用的符号键  
    - speller  
    - punctuator  # 否则「，。」就会由此上屏  
    - selector  
    - navigator  
    - express\_editor  
  segmentors:  
    - abc\_segmentor  
    - punct\_segmentor  
    - fallback\_segmentor  
  translators:  
    - echo\_translator  
    - punct\_translator  
    - table\_translator  
  
translator:  
  dictionary: hello  
  
punctuator:  
  import\_preset: default  
  
key\_binder:  
  bindings:             # 每条定义包含条件、接收按键（IBus规格的键名，可加修饰符，如「Control+Return」）、发送按键  
  
    - when:   paging    # 仅当已发生向后换页时，  
      accept: comma     # 将「逗号」键……  
      send:   Page\_Up   # 关联到「向前换页」；于是 navigator 将收到一发 Page\_Up  
  
    - when:   has\_menu  # 只要有候选字即满足条件  
      accept: period  
      send:   Page\_Down

**【二】修炼之道**

与【大家好】这个方案不同。以下一组示例，主要演示如何活用符号键盘，以及罗马字转写式输入。

**改造键盘**

莫以为【大家好】是最最简单的输入方案。码表式输入法，不如「键盘式」输入法来得简单明快！

用 punctuator 这一套组件，就可实现一款键盘输入法：

# Rime schema  
# encoding: utf-8  
  
schema:  
  schema\_id: numbers  
  name: 数字之道  
  version: "1"  
  
engine:  
  processors:  
    - punctuator  
    - express\_editor  
  segmentors:  
    - punct\_segmentor  
  translators:  
    - punct\_translator  
  
punctuator:  
  half\_shape: &symtable  
    "1" : 一  
    "2" : 二  
    "3" : 三  
    "4" : 四  
    "5" : 五  
    "6" : 六  
    "7" : 七  
    "8" : 八  
    "9" : 九  
    "0" : 〇  
    "s" : 十  
    "b" : 百  
    "q" : 千  
    "w" : 万  
    "n" : 年  
    "y" : [ 月, 元, 亿 ]  
    "r" : 日  
    "x" : 星期  
    "j" : 角  
    "f" : 分  
    "z" : [ 之, 整 ]  
    "d" : 第  
    "h" : 号  
    "." : 点  
  full\_shape: \*symtable

对，所谓「键盘输入法」，就是按键和字直接对应的输入方式。

这次，不再写 punctuator/import\_preset 这项，而是自定义了一套符号表。

鸹！原来 punctuator 不单可以用来打出标点符号；还可以重定义空格以及全部 94 个可打印 ASCII 字符（码位 0x20 至 0x7e）。

在符号表代码里，用对应的ASCII字符表示按键。记得这些按键字符要放在引号里面，YAML 纔能够正确解析喔。

示例代码表演了两种符号的映像方式：一对一及一对多。一对多者，按键后符号不会立即上屏，而是……嘿嘿，自己体验吧 :-)

关于代码里 symtable 的一点解释：

这是YAML的一种语法，&symtable 叫做「锚点标签」，给紧随其后的内容起个名字叫 symtable； \*symtable 则相当于引用了symtable 所标记的那段内容，从而避免重复。

Rime 里的符号有「全角」、「半角」两种状态。本方案里暂不作区分，教 half\_shape、full\_shape 使用同一份符号表。

**大写数字键盘**

灵机一动，不如利用「全、半角」模式来区分「大、小写」中文数字！

schema:  
  # ...  
  version: "2"  
  
switches:  
  - name: full\_shape  
    states: [ 小写, 大写 ]  
  
# ...

先来定义状态开关：0态改「半角」为「小写」，1态改「全角」为「大写」。

这样一改，再打开「方案选单」，方案「数字之道」底下就会多出个「小写→大写」的选项，每选定一次、状态随之反转一次。

接着给 half\_shape、full\_shape 定义不同的符号表：

punctuator:  
  half\_shape:  
    "1" : 一  
    "2" : 二  
    "3" : 三  
    "4" : 四  
    "5" : 五  
    "6" : 六  
    "7" : 七  
    "8" : 八  
    "9" : 九  
    "0" : 〇  
    "s" : 十  
    "b" : 百  
    "q" : 千  
    "w" : 万  
    "n" : 年  
    "y" : [ 月, 元, 亿 ]  
    "r" : 日  
    "x" : 星期  
    "j" : 角  
    "f" : 分  
    "z" : [ 之, 整 ]  
    "d" : 第  
    "h" : 号  
    "." : 点  
  full\_shape:  
    "1" : 壹  
    "2" : 贰  
    "3" : 参  
    "4" : 肆  
    "5" : 伍  
    "6" : 陆  
    "7" : 柒  
    "8" : 捌  
    "9" : 玖  
    "0" : 零  
    "s" : 拾  
    "b" : 佰  
    "q" : 仟  
    "w" : 万  
    "n" : 年  
    "y" : [ 月, 圆, 亿 ]  
    "r" : 日  
    "x" : 星期  
    "j" : 角  
    "f" : 分  
    "z" : [ 之, 整 ]  
    "d" : 第  
    "h" : 号  
    "." : 点

哈，调出选单切换一下大小写，输出的字全变样！酷。

但是要去选单切换，总不如按下 Shift 就全都有了：

punctuator:  
  half\_shape:  
    # ... 添加以下这些  
    "!" : 壹  
    "@" : 贰  
    "#" : 参  
    "$" : [ 肆, ￥, "$", "€", "£" ]  
    "%" : [ 伍, 百分之 ]  
    "^" : 陆  
    "&" : 柒  
    "\*" : 捌  
    "(" : 玖  
    ")" : 零  
    "S" : 拾  
    "B" : 佰  
    "Q" : 仟  
    "Y" : 圆

于是在「小写」态，只要按 Shift + 数字键即可打出大写数字。

用了几下，发现一处小小的不满意：敲 $ 这个键，可选的符号有五个之多。想要打出殴元、英镑符号只得多敲几下 $ 键使想要的符号高亮；但是按上、下箭头键并没有效果，按符号前面标示的数字序号，更是不仅上屏了错误的符号、还多上屏一个数字——

这反映出两个问题。一是 selector 组件缺席使得选字、移动选字光目标动作未得到响应。立即加上：

# ...  
  
engine:  
  processors:  
    - punctuator  
    - selector        # 加在这里  
    - express\_editor  
  # ...

因为要让 punctuator 来转换数字键，所以 selector 得放在他后头。

好。二一个问题还在：无法用数字序号选字。为解决这个冲突，改用闲置的字母键来选字：

# ...  
  
menu:  
  alternative\_select\_keys: "acegi"

完工。

**罗马字之道**

毕竟，键盘上只有47个字符按键、94个编码字符，对付百十个字还管使。可要输入上千个常用汉字，嫌键盘式输入的编码空间太小，必得采用多字符编码。

罗马字，以拉丁字母的特定排列作为汉语音节的转写形式。一个音节代表一组同音字，再由音节拼写组合成词、句。

凡此单字（音节）编码自然连用而生词、句的输入法，皆可用 r10n\_translator 组件完成基于音节码切分的智能词句转换。r10n 为romanization 的简写。但不限于「拼音」、「注音」、「双拼」、「粤拼」等一族基于语音编码的输入法：形式相似者，如「速成」，虽以字形为本，亦可应用。

现在来把【数字之道】改成拼音→中文数字的变换。

schema:  
  schema\_id: numbers  
  name: 数字之道  
  version: "3"  
  
engine:  
  processors:  
    - speller  
    - punctuator  
    - selector  
    - express\_editor  
  segmentors:  
    - abc\_segmentor  
    - punct\_segmentor  
  translators:  
    - punct\_translator  
    - r10n\_translator  
  
translator:  
  dictionary: numbers  
  
punctuator:  
  half\_shape: &symtable  
    "!" : 壹  
    "@" : 贰  
    "#" : 参  
    "$" : [ 肆, ￥, "$", "€", "£" ]  
    "%" : [ 伍, 百分之 ]  
    "^" : 陆  
    "&" : 柒  
    "\*" : 捌  
    "(" : 玖  
    ")" : 零  
    "S" : 拾  
    "B" : 佰  
    "Q" : 仟  
    "W" : 万  
    "N" : 年  
    "Y" : [ 月, 圆, 亿 ]  
    "R" : 日  
    "X" : 星期  
    "J" : 角  
    "F" : 分  
    "Z" : [ 之, 整 ]  
    "D" : 第  
    "H" : 号  
    "." : 点  
  full\_shape: \*symtable

符号表里，把小写字母、数字键都空出来了。小写字母用来拼音，数字键用来选重。重点是本次用了 r10n\_translator 这组件。与table\_translator 相似，该组件与 translator/dictionary 指名的词典相关联。

编制词典：

# Rime dictionary  
# encoding: utf-8  
  
---  
name: numbers  
version: "1"  
sort: by\_weight  
use\_preset\_vocabulary: true  
...  
  
一       yi  
二       er  
三       san  
四       si  
五       wu  
六       liu  
七       qi  
八       ba  
九       jiu  
〇       ling  
零       ling  
十       shi  
百       bai  
千       qian  
万       wan  
亿       yi  
年       nian  
月       yue  
日       ri  
星       xing  
期       qi  
时       shi  
分       fen  
秒       miao  
元       yuan  
角       jiao  
之       zhi  
整       zheng  
第       di  
号       hao  
点       dian  
是       shi

※注意： **不要** 从网页复制以上代码到实作的词典文件！因为网页里制表符被转换成空格从而不符合 Rime 词典要求的格式。

码表里给出了一个「示例」规格的小字集。其中包含几组重码字。

要诀 sort: by\_weight 意图是不以码表的顺序排列重码字，而是比较字频。那字频呢？没写出来。

要诀 use\_preset\_vocabulary: true 用在输入方案需要支持输入词组、而码表中词组相对匮乏时。编译输入方案期间引入 Rime 预设的【八股文】词汇——及词频资料！这就是码表中未具字频的原因。

使用【八股文】，要注意码表所用的字形是否与该词汇表一致。八股文的词汇及词频统计都遵照 opencc 繁体字形标准。

如果缺少单字的编码定义，自然也无法导入某些词汇。所以本方案只会导入这个数字「小字集」上的词汇。

**用拼写运算定义简码**

如今有了一款专门输入数字的拼音输入法。比一比升阳拼音、朙月拼音和地球拼音，还有哪里不一样？

很快我发现敲 xingqiwu 或 xingqiw 都可得到来自【八股文】的词组「星期五」，这很好。可是敲 xqw 怎会不中呢？

原来 r10n\_translator 罗马字中译的方法是，将输入码序列切分为音节表中的拼写形式，再按音节查词典。不信你找本词典瞧瞧，是不是按完整的拼音（注音）编排的。Rime 词典也一样。并没有 xqw 这样的检索码。

现在我要用 Rime 独门绝活「拼写运算」来定义一种「音序查字法」。令 x 作 xing 的简码，q 作数字之道所有音节中起首为 q 者的简码，即略代音节 qi 与 qian。

「汉语拼音」里还有三个双字母的声符，zh, ch, sh 也可做简码。

添加拼写运算规则：

schema:  
  # ...  
  version: "4"  
  
#...  
  
speller:  
  algebra:  
    - 'abbrev/^([a-z]).+$/$1/'  
    - 'abbrev/^([zcs]h).+$/$1/'

如此 Rime 便知，除了码表里那些拼音，还有若干简码也是行得通的拼写形式。再输入 xqw，Rime 将他拆开 x'q'w，再默默对应到音节码xing'qi'wan、xing'qi'wu、xing'qian'wan 等等，一翻词典就得到了一个好词「星期五」，而其他的组合都说不通。

现在有无有悟到，罗马字转写式输入法与码表式输入法理念上的不同？

哈，做中了。试试看 sss,sss,sssss,sssss

却好像不是我要的「四是四，十是十，十四是十四，四十是四十」……

好办。如果某些词汇在方案里很重要，【八股文】又未收录，那么，请添加至码表：

---  
name: numbers  
version: "2"  
sort: by\_weight  
use\_preset\_vocabulary: true  
...  
  
# ...  
  
四是四  
十是十  
十四是十四  
四十是四十

善哉。演示完毕。当然休想就此把 Rime 全盘掌握了。一本《指南书》，若能让读者入门，我止说「善哉〜」

再往后，就只有多读代码，纔能见识到各种新颖、有趣的玩法。

**【三】最高武艺**

〔警告〕最后这部戏，对智力、技术功底的要求不一般。如果读不下去，不要怪我、不要怀疑自己的智商！

即使跳过本节书也无妨，只是不可忽略了下文《关于调试》这一节！（重要哇……）

请检查是否：

* ※ 已将前两组实例分析透彻
* ※ 学习完了《拼写运算》
* ※ 知道双拼是神码
* ※ 预习 Rime 预设输入方案之【朙月拼音】

设计一款【智能ABC双拼】输入方案做练习！

# Rime schema  
# encoding: utf-8  
  
schema:  
  schema\_id: double\_pinyin\_abc  # 专有的方案标识  
  name: 智能ABC双拼  
  version: "0.9"  
  author:  
    - 佛振 <chen.sst@gmail.com>  
  description: |  
    朙月拼音，兼容智能ABC双拼方案。  
  
switches:  
  - name: ascii\_mode  
    reset: 0  
    states: [ 中文, 西文 ]  
  - name: full\_shape  
    states: [ 半角, 全角 ]  
  - name: simplification  
    states: [ 汉字, 汉字 ]  
  
engine:  
  processors:  
    - ascii\_composer  
    - recognizer  
    - key\_binder  
    - speller  
    - punctuator  
    - selector  
    - navigator  
    - express\_editor  
  segmentors:  
    - ascii\_segmentor  
    - matcher  
    - abc\_segmentor  
    - punct\_segmentor  
    - fallback\_segmentor  
  translators:  
    - echo\_translator  
    - punct\_translator  
    - r10n\_translator  
    - reverse\_lookup\_translator  
  filters:  
    - simplifier  
    - uniquifier  
  
speller:  
  alphabet: zyxwvutsrqponmlkjihgfedcba  # 呃，倒背字母表完全是个人喜好  
  delimiter: " '"  # 隔音符号用「'」；第一位的空白用来自动插入到音节边界处  
  algebra:  # 拼写运算规则，这个纔是实现双拼方案的重点。写法有很多种，当然也可以把四百多个音节码一条一条地列举  
    - erase/^xx$/             # 码表中有几个拼音不明的字，编码成xx了，消灭他  
    - derive/^([jqxy])u$/$1v/  
    - xform/^zh/A/            # 替换声母键，用大写以防与原有的字母混淆  
    - xform/^ch/E/  
    - xform/^sh/V/  
    - xform/^([aoe].\*)$/O$1/  # 添上固定的零声母o，先标记为大写O  
    - xform/ei$/Q/            # 替换韵母键  
    - xform/ian$/W/           # ※2  
    - xform/er$|iu$/R/        # 对应两种韵母的；音节er现在变为OR了  
    - xform/[iu]ang$/T/       # ※1  
    - xform/ing$/Y/  
    - xform/uo$/O/  
    - xform/uan$/P/           # ※3  
    - xform/i?ong$/S/  
    - xform/[iu]a$/D/  
    - xform/en$/F/  
    - xform/eng$/G/  
    - xform/ang$/H/           # 检查一下在此之前是否已转换过了带介音的ang；好，※1处有了  
    - xform/an$/J/            # 如果※2、※3还无有出现在上文中，应该把他们提到本行之前  
    - xform/iao$/Z/           # 对——像这样让iao提前出场  
    - xform/ao$/K/  
    - xform/in$|uai$/C/       # 让uai提前出场  
    - xform/ai$/L/  
    - xform/ie$/X/  
    - xform/ou$/B/  
    - xform/un$/N/  
    - xform/[uv]e$|ui$/M/  
    - xlit/QWERTYOPASDFGHJKLZXCVBNM/qwertyopasdfghjklzxcvbnm/  # 最后把双拼码全部变小写  
  
translator:  
  dictionary: luna\_pinyin     # 与【朙月拼音】共享词典  
  prism: double\_pinyin\_abc    # prism 要以本输入方案的名称来命名，以免把朙月拼音的拼写映射表覆盖掉  
  preedit\_format:             # 这段代码用来将输入的双拼码反转为全拼显示；待见双拼码的可以把这段拿掉  
    - xform/o(\w)/0$1/        # 零声母先改为0，以方便后面的转换  
    - xform/(\w)q/$1ei/       # 双拼第二码转换为韵母  
    - xform/(\w)n/$1un/       # 提前转换双拼码 n 和 g，因为转换后的拼音里就快要出现这两个字母了，那时将难以分辨出双拼码  
    - xform/(\w)g/$1eng/      # 当然也可以采取事先将双拼码变为大写的办法来与转换过的拼音做区分，可谁让我是高手呢  
    - xform/(\w)w/$1ian/  
    - xform/([dtnljqx])r/$1iu/  # 对应多种韵母的双拼码，按搭配的声母做区分（最好别用排除式如 [^o]r 容易出状况）  
    - xform/0r/0er/             # 另一种情况，注意先不消除0，以防后面把e当作声母转换为ch  
    - xform/([nljqx])t/$1iang/  
    - xform/(\w)t/$1uang/       # 上一行已经把对应到 iang 的双拼码 t 消灭，于是这里不用再列举相配的声母  
    - xform/(\w)y/$1ing/  
    - xform/([dtnlgkhaevrzcs])o/$1uo/  
    - xform/(\w)p/$1uan/  
    - xform/([jqx])s/$1iong/  
    - xform/(\w)s/$1ong/  
    - xform/([gkhaevrzcs])d/$1ua/  
    - xform/(\w)d/$1ia/  
    - xform/(\w)f/$1en/  
    - xform/(\w)h/$1ang/  
    - xform/(\w)j/$1an/  
    - xform/(\w)k/$1ao/       # 默默检查：双拼码 o 已经转换过了  
    - xform/(\w)l/$1ai/  
    - xform/(\w)z/$1iao/  
    - xform/(\w)x/$1ie/  
    - xform/(\w)b/$1ou/  
    - xform/([nl])m/$1ve/  
    - xform/([jqxy])m/$1ue/  
    - xform/(\w)m/$1ui/  
    - "xform/(^|[ '])a/$1zh/"  # 复原声母，音节开始处的双拼字母a改写为zh；其他位置的才真正是a  
    - "xform/(^|[ '])e/$1ch/"  
    - "xform/(^|[ '])v/$1sh/"  
    - xform/0(\w)/$1/          # 好了，现在可以把零声母拿掉啦  
    - xform/([nljqxy])v/$1ü/   # 这样纔是汉语拼音 :-)  
  
reverse\_lookup:  
  dictionary: cangjie5  
  prefix: "`"  
  tips: 〔仓颉〕  
  preedit\_format:  
    - "xlit|abcdefghijklmnopqrstuvwxyz|日月金木水火土竹戈十大中一弓人心手口尸廿山女田难卜符|"  
  comment\_format:  
    - xform/([nl])v/$1ü/  
  
punctuator:  
  import\_preset: default  
  
key\_binder:  
  import\_preset: default  
  
recognizer:  
  import\_preset: default  
  patterns:  
    reverse\_lookup: "`[a-z]\*$"

完毕。

这是一道大题。通过改造拼写法而创作出新的输入方案。

**关于调试**

如此复杂的输入方案，很可能需要反复调试方可达到想要的结果。

请于试验时及时查看日志中是否包含错误信息。日志文件位于：

* 【中州韵】 /tmp/rime.ibus.\*
* 【小狼毫】 %TEMP%\rime.weasel.\*
* 【鼠须管】 $TMPDIR/rime.squirrel.\*
* 各发行版的早期版本 用户文件夹/rime.log

按照日志的级别分为 INFO / 信息、WARNING / 警告、ERROR / 错误。 后两类应重点关注，如果新方案部署后不可用或输出与设计不一致，原因可能在此。

没有任何错误信息，就是不好使，有可能是码表本身的问题，比如把码表中文字和编码两列弄颠倒了——Rime 等你输入由汉字组成的编码，然而键盘没有可能做到这一点（否则也不再需要输入法了）。

后续有计划为输入方案创作者开发名为「拼写运算调试器」的工具，能够较直观地看到每一步拼写运算的结果。有助于定义双拼这样大量使用拼写运算的方案。

**东风破**

「东风破早梅，向暖一枝开。」

构想在 Rime 输入软件完善后、能够链接汉字字形、音韵、输入法爱好者的共同兴趣，形成稳定的使用者社羣，搭建一个分享知识的平台。

而【东风破】，定义为 Rime 输入方案仓库，则是分享 Rime 输入方案的平台。

Rime 是一款强调个性的输入法。

Rime 不要定义输入法应当是哪个样、而要定义输入法可以玩出哪些花样。

Rime 不可能通过预设更多的输入方案来满足玩家的需求；真正的玩家一定有一般人想不到的高招。

未来一定会有，【东风破】，让用家轻松地找到最新、最酷、最适合自己的 Rime 输入方案。