RimeWithSchemata

Jump to bottom

davix edited this page on Jul 18, 2018 · 15 revisions

> Rime 輸入方案設計書

Rime with Schemata

Rime 輸入方案創作者的第一本參考書

※ 佛振 chen.sst@gmail.com 修訂於 2013年5月4日

Rime 輸入方案

作者的一點主張：選用適合自己的輸入法。

來折騰 Rime 的同學，我只能說，您對輸入法的追求比較與衆不同啦！

當然，因爲您理想中的輸入方式千奇百怪、也許從沒有人那樣玩過，所以不可能在那種勾勾選選的介面上做得出來；需要親手來創作——

Rime 輸入方案！

Rime 是啥

Rime 不是一種輸入法。是從各種常見鍵盤輸入法中提煉出來的抽象的輸入算法框架。因爲 Rime 涵蓋了大多數輸入法的「共性」，所以在不同的設定下，Rime 可化身爲不同的輸入法用來打字。

Rime 輸入方案又是啥

要讓 Rime 實現某種具體輸入法的功能，就需要一些數據來描述這種輸入法以何種形式工作。即，定義該輸入法的「個性」。

如「漢語拼音」、「注音」、「倉頡碼」、「五筆字型」，這些方法可憑藉 Rime 提供的通用設施、給定不同的工作參數來實作。以本文介紹的規格寫成一套套的配方，就是 Rime 輸入方案。

爲啥這麼繁？

一鍵就搞掂，必然選項少，功能單一。不好玩。

輸入法程序一寫兩三年，也許還不夠火候；花兩三個小時來讀入門書，已是輸入法專業速成班。

準備開工

Rime 是跨平臺的輸入法軟件，Rime 輸入方案可通用於以下發行版：

- 【中州韻】 ibus-rime → Linux
- 【小狼毫】 Weasel → Windows
- 【鼠鬚管】 Squirrel → Mac OS X

取得適合你系統的最新版 Rime 輸入法，印一份《指南書》，準備開工了！

Rime with Text Files

文本爲王。 Rime 的配置文件、輸入方案定義及詞典文件，均爲特定格式的文本文檔。 因此，一款夠專業的 **文本編輯器** ，是設計 Rime 輸入方案的必備工具。

Rime 中所有文本文檔，均要求以 UTF-8 編碼，並建議使用 UNIX 換行符（LF）。

鑑於一些文本編輯器會爲 UTF-8 編碼的文件添加 BOM 標記，爲防止誤將該字符混入文中， 莫要從文件的第一行開始正文，而請在該行行首以 # 記號起一行註釋，如：

```
# Rime default settings

# Rime schema: My First Cool Schema

# Rime dictionary: Lingua Latina
```

也可繼續以註釋行寫下方案簡介、碼表來源、製作者、修訂記錄等信息，再切入正文。

必知必會

Rime 輸入法中，多用擴展名爲「`.yaml`」的文本文檔，這是以一種可讀性高的數據描述語言——YAML 寫成。

請訪問 http://yaml.org/ 瞭解 YAML 文檔格式。 下文只對部分語法作簡要說明，而將重點放在對語義的解讀上面。

Rime 輸入方案亦會用到「正則表達式」實現一些高級功能。

輸入方案設計者需要掌握這份文檔所描述的 Perl 正則表達式語法：

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

Rime 中的數據文件分佈及作用

除程序文件以外，Rime 還包括多種數據文件。 這些數據文件存在於以下位置：

共享資料夾

- 【中州韻】 `/usr/share/rime-data/`
- 【小狼毫】 `"安裝目錄\data"`
- 【鼠鬚管】 `"/Library/Input Methods/Squirrel.app/Contents/SharedSupport/"`

用戶資料夾

- 【中州韻】 `~/.config/ibus/rime/` （0.9.1 以下版本爲 `~/.ibus/rime/`）
- 【小狼毫】 `"%APPDATA%\Rime"`
- 【鼠鬚管】 `~/Library/Rime/`

共享資料夾包含預設輸入方案的源文件。 這些文件屬於 Rime 所發行軟件的一部份，在訪問權限控制較嚴格的系統上對用戶是只讀的，因此謝絕軟件版本更新以外的任何修改—— 一旦用戶修改這裏的文件，很可能影響後續的軟件升級或在升級時丟失數據。

在「部署 Rime」操作時，將用到這裏的輸入方案源文件、並結合用戶定製的內容來編譯預設輸入方案。

用戶資料夾則包含爲用戶準備的內容，如：

- 〔全局設定〕 `default.yaml`
- 〔發行版設定〕 `weasel.yaml`
- 〔預設輸入方案副本〕 `<方案標識>.schema.yaml`
- ※〔安裝信息〕 `installation.yaml`
- ※〔用戶狀態信息〕 `user.yaml`

編譯輸入方案所產出的二進制文件：

- 〔Rime 棱鏡〕 `<方案標識>.prism.bin`

- 〔Rime 固態詞典〕 `<詞典名>.table.bin`
- 〔Rime 反查詞典〕 `<詞典名>.reverse.bin`

記錄用戶寫作習慣的文件：

- ※〔用戶詞典〕 `<詞典名>.userdb.kct`
- ※〔用戶詞典快照〕 `<詞典名>.userdb.txt` 、 `<詞典名>.userdb.kct.snapshot` 見於同步文件夾

以及用戶自己設定的：

- ※〔用戶對全局設定的定製信息〕 `default.custom.yaml`
- ※〔用戶對預設輸入方案的定製信息〕 `<方案標識>.custom.yaml`
- ※〔用戶自製輸入方案〕及配套的詞典源文件

註：以上標有 ※ 號的文件，包含用戶資料，您在清理文件時要注意備份！

詳解輸入方案

方案定義

一套輸入方案，通常包含「方案定義」和「詞典」文件。

方案定義，命名爲 `<方案標識>.schema.yaml` ，是一份包含輸入方案配置信息的 YAML 文檔。

文檔中需要有這樣一組方案描述：

```
# 以下代碼片段節選自 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 預設的拼音輸入方案。
```

首先來爲方案命名。 `schema/name` 字段是顯示在〔方案選單〕中的名稱。

然後——重點是——要定一個在整個 Rime 輸入法中唯一的「方案標識」，即 `schema/schema_id` 字段的內容。 方案標識由小寫字母、數字、下劃線構成。 僅於輸入法內部使用，且會構成方案定義文件名的一部分，因此爲了兼容不同的文件系統，不要用大寫字母、漢字、空格和其他符號做方案標識。 一例：這款名爲【朙月拼音】的輸入方案，方案標識爲「 `luna_pinyin` 」。

方案如做升級，通過版本號（ `schema/version` ）來區分相互兼容的新舊版本。

版本號——以「.」分隔的整數（或文字）構成的字符串。

如下都是版本號常見的形式：

```
"1"      # 最好加引號表明是字符串！
"1.0"    # 最好加引號表明是字符串！
"0.9.8"
"0.9.8.custom.86427531"  # 這種形式是經過用戶自定義的版本；自動生成
```

然而，若對方案的升級會導致原有的用戶輸入習慣無法在新的方案中繼續使用，則需要換個新的方案標識。

比如【倉頡五代】之於【倉頡三代】、【五筆98】之於【五筆86】，其實已是互不兼容的輸入法。

`schema/author` ——列出作者和主要貢獻者，格式爲文字列表：

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

`schema/description` ——對方案作簡要介紹的多行文字。

以上 `schema/schema_id`、`schema/version` 字段用於在程序中識別輸入方案， 而 `schema/name`、`schema/author`、`schema/description` 則主要是展示給用戶的信息。

除方案描述外，方案定義文件中還包含各種功能設定，控制着輸入法引擎的工作方式。

輸入法引擎與功能組件

論 Rime 輸入法的工作流程：

> 按鍵消息→後臺「服務」→分配給對應的「會話」→由「方案選單」或「輸入引擎」處理……

這裏要點是，有會話的概念：多窗口、多線操作嘛，你懂得的，同時與好幾位 MM 聊天時，有無有好幾組會話。 每一組會話中都有一部輸入引擎完成按鍵序列到文字的變換。

Rime 可以在不同會話裏使用不同輸入方案。因爲有「方案選單」。 方案選單本身可響應一些按鍵。但由於他不會寫字的緣故，更多時候要把按鍵遞給同一會話中的「輸入引擎」繼續處理。 方案選單的貢獻，就是給用戶一個便捷的方案切換介面，再把用戶挑中的輸入方案加載到輸入引擎。

再論輸入引擎的工作流程：

> 加載輸入方案、預備功能組件；各就位之後就進入處理按鍵消息、處理按鍵消息……的循環。

響應各種按鍵、產生各類結果的工作，由不同的功能組件分擔。

好，看代碼：

```
# luna_pinyin.schema.yaml
# ...

engine:                   # 輸入引擎設定，即掛接組件的「處方」
  processors:             # 一、這批組件處理各類按鍵消息
    - ascii_composer      # ※ 處理西文模式及中西文切換
    - recognizer          # ※ 與 matcher 搭配，處理符合特定規則的輸入碼，如網址、反查等
    - key_binder          # ※ 在特定條件下將按鍵綁定到其他按鍵，如重定義逗號、句號爲候選翻
頁鍵
    - speller             # ※ 拼寫處理器，接受字符按鍵，編輯輸入碼
    - punctuator          # ※ 句讀處理器，將單個字符按鍵直接映射爲文字符號
    - selector            # ※ 選字處理器，處理數字選字鍵、上、下候選定位、換頁鍵
    - navigator           # ※ 處理輸入欄內的光標移動鍵
    - express_editor      # ※ 編輯器，處理空格、回車上屏、回退鍵等
  segmentors:             # 二、這批組件識別不同內容類型，將輸入碼分段
    - ascii_segmentor     # ※ 標識西文段落
    - matcher             # ※ 標識符合特定規則的段落，如網址、反查等
    - abc_segmentor       # ※ 標識常規的文字段落
    - punct_segmentor     # ※ 標識句讀段落
    - fallback_segmentor  # ※ 標識其他未標識段落
  translators:            # 三、這批組件翻譯特定類型的編碼段爲一組候選文字
    - echo_translator     # ※ 沒有其他候選字時，回顯輸入碼
    - punct_translator    # ※ 轉換標點符號
    - script_translator   # ※ 腳本翻譯器，用於拼音等基於音節表的輸入方案
    - reverse_lookup_translator  # ※ 反查翻譯器，用另一種編碼方案查碼
  filters:                # 四、這批組件過濾翻譯的結果
    - simplifier          # ※ 繁簡轉換
    - uniquifier          # ※ 過濾重複的候選字，有可能來自繁簡轉換
```

註：除示例代碼中引用的組件外，尚有

```
- fluid_editor      # ※ 句式編輯器，用於以空格斷詞、回車上屏的【注音】、【語句流】等輸入方
案，替換 express_editor，也可以寫作 fluency_editor
- chord_composer    # ※ 和絃作曲家或曰並擊處理器，用於【宮保拼音】等多鍵並擊的輸入方案
- table_translator  # ※ 碼表翻譯器，用於倉頡、五筆等基於碼表的輸入方案，替換
script_translator
```

輸入引擎把完成具體功能的邏輯拆分爲可裝卸、組合的部件。 「加載」輸入方案，即按該處方掛接所需的功能組件、令這些組件從輸入方案定義中加載各自的設定、準備各司其職。 而他們接下來要完成的作業，由引擎收到的一份按鍵消息引發。

理解 Processors

輸入引擎，作爲整體來看，以按鍵消息爲輸入，輸出包括三部分：

- 一是對按鍵消息的處理結果：操作系統要一個結果、這按鍵、輸入法接是不接？

- 二是暫存於輸入法、尚未完成處理的內容，會展現在輸入法候選窗中。
- 三是要「上屏」的文字，並不是每按一鍵都有輸出。通常中文字都會伴隨「確認」動作而上屏，有些按鍵則會直接導致符號上屏，而這些還要視具體場景而定。

那麼第一類功能組件 `processor`s，就是比較籠統地、起着「處理」按鍵消息的作用。

按鍵消息依次送往列表中的 `processor` ，由他給出對按鍵的處理意見：

- 或曰「收」、即由 Rime 響應該按鍵；
- 或曰「拒」、回禀操作系統 Rime 不做響應、請對按鍵做默認處理；
- 或曰這個按鍵我不認得、請下一個 `processor` 繼續看。

優先級依照 `processors` 列表順序排定，接收按鍵者會針對按鍵消息做處理。

雖然看起來 `processor` 通過組合可以承擔引擎的全部任務，但爲了將邏輯繼續細分、Rime 又爲引擎設置了另外三類功能組件。這些組件都可以訪問引擎中的數據對象——輸入上下文，並將各自所做處理的階段成果存於其中。

`processor` 最常見的處理，便是將按鍵所產生的字符記入上下文中的「輸入碼」序列。 當「輸入碼」發生變更時，下一組組件 `segmentor`s 開始一輪新的作業。

理解 Segmentors

Rime 可將文字、數字、符號等不同內容連續輸入，此時需要識別不同格式的輸入碼、將輸入碼分成若干段分而治之。 這通過數輪代碼段劃分操作完成。每一輪操作中、一衆 `segmentor`s 分別給出起始於某一處、符合特定格式的代碼段，識別到的最長代碼段成爲本輪劃分的結果，而給出這一劃分的一個或多個 `segmentor` 組件則可爲該代碼段打上「類型標籤」；從這一新代碼段的結束位置，開始下一輪劃分，直到整個輸入碼序列劃分完畢。

舉例來說，【朙月拼音】中，輸入碼 `2012nian\` ，劃分爲三個編碼段： `2012` （貼`number`標籤）、 `nian` （貼`abc`標籤）、 `\` （貼`punct`標籤）。

那些標籤是初步劃分後判定的類型，也可能有一個編碼段貼多個標籤的情況。下一個階段中，`translator`s 會把特定類型的編碼段翻譯爲文字。

理解 Translators

顧名思義， `translator` 完成由編碼到文字的翻譯。但有幾個要點：

- 一是翻譯的對象是劃分好的一個代碼段。
- 二是某個 `translator` 組件往往只翻譯具有特定標籤的代碼段。
- 三是翻譯的結果可能有多條，每條結果成爲一個展現給用戶的候選項。
- 四是代碼段可由幾種 `translator` 分別翻譯、翻譯結果按一定規則合併成一列候選。
- 五是候選項所對應的編碼未必是整個代碼段。用拼音敲一個詞組時，詞組後面繼續列出單字候選，即是此例。

雙目如探針般進入內存查看，發現翻譯的結果呈現這種形式：

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

輸入串劃分爲多個代碼段、每段代碼又可具有多組翻譯結果；取各代碼段的首選結果連接起來，就是預備上屏的文字「`2012年、`」。

且將以上所示的數據稱爲「作文」。這是一篇未定稿（未搞定）的作文，輸入法介面此時顯示預備上屏的文字「`2012年、`」，並列出最末一個代碼段上的候選「`、`」及「`\`」以供選擇。

有兩款主力 `translator` 完成主要文字內容的翻譯，其實現的方式很不一樣：

- `script_translator` 也叫做 `r10n_translator` 修煉羅馬字分析法，以「固定音節表」爲算法的基礎，識別輸入碼的音節構成，推敲排列組合、完成遣詞造句。
- `table_translator` 修煉傳承自上世紀的碼表功夫，基於規則的動態碼表，構成編碼空間內一個開放的編碼集合。

拼音、注音、方言拼音，皆是以固定音節表上的拼寫排列組合的方式產生編碼，故適用羅馬字分析法。 倉頡、五筆字型這類則是傳統的碼表輸入法。

如果以碼表方式來寫拼音輸入方案，是怎樣的效果呢？雖然仍可完成輸入，但無法完全實現支持簡拼、模糊拼音、使用隔音符號的動態調頻、智能語句等有用的特性。

反之，以羅馬字方式使用碼表輸入法，則無法實現定長編碼頂字上屏、按編碼規則構詞等功能。在 Rime 各發行版預設輸入方案中，有一款「速成」輸入方案，即是以 `script_translator` 翻譯倉頡碼，實現全、簡碼混合的語句輸入。

概括起來，這是兩種構造新編碼的方式：羅馬字式輸入方案以一組固定的基本音節碼創造新的組合而構詞，而碼表式輸入方案則以一定碼長爲限創造新的編碼映射而構詞。

理解 Filters

上一步已經收集到各個代碼段的翻譯結果，當輸入法需要在介面呈現一頁候選項時，就從最末一個代碼段的結果集中挑選、直至取夠方案中設定的頁最大候選數。

每從結果集選出一條字詞、會經過一組 `filter`s 過濾。多個 `filter` 串行工作，最終產出的結果進入候選序列。

`filter` 可以：

- 改裝正在處理的候選項，修改某些屬性值：簡化字、火星文、菊花文有無有？過時了！有 Rime，你對文字的想象力終於得救

- 消除當前候選項，比如檢測到重複（由不同 `translator` 產生）的候選條目
- 插入新的候選項，比如根據已有條目插入關聯的結果
- 修改已有的候選序列

碼表與詞典

詞典是 `translator` 的參考書。

他往往與同名輸入方案配套使用，如拼音的詞典以拼音碼查字，倉頡的詞典以倉頡碼查字。但也可以由若干編碼屬於同一系統的輸入方案共用，如各種雙拼方案，都使用和拼音同樣的詞典，於是不僅復用了碼表數據，也可共享用戶以任一款此系列方案錄入的自造詞（仍以碼表中的形式即全拼編碼記錄）。

Rime 的詞典文件，命名爲 `<詞典名>.dict.yaml`，包含一份碼表及對應的規則說明。 詞典文件的前半部份爲一份 YAML 文檔：

```
# 注意這裏以 --- ... 分別標記出 YAML 文檔的起始與結束位置
# 在 ... 標記之後的部份就不會作 YAML 文檔來解讀

---
name: luna_pinyin
version: "0.9"
sort: by_weight
use_preset_vocabulary: true
...
```

解釋：

- `name`: 詞典名，內部使用，命名原則同「方案標識」；可以與配套的輸入方案標識一致，也可不同；
- `version`: 管理詞典的版本，規則同輸入方案定義文件的版本號；
- `sort`: 詞條初始排序方式，可選填 `by_weight`（按詞頻高低排序）或 `original`（保持原碼表中的順序）；
- `use_preset_vocabulary`: 填 `true` 或 `false`，選擇是否導入預設詞彙表【八股文】。

碼表，定義了輸入法中編碼與文字的對應關係。

碼表位於詞典文件中 YAML 結束標記之後的部份。 其格式爲以製表符分隔的值（TSV），每行定義一條「文字 – 編碼」的對應關係：

```
# 單字
你	ni
我	wo
的	de	99%
的	di	1%
```

```
地	de	10%
地	di	90%
目	mu
好	hao

# 詞組
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* 從網頁複製以上代碼到實作的詞典文件！因爲網頁裏製表符被轉換成空格從而不符合 Rime 詞典要求的格式。

碼表部份，除了以上格式的編碼行，還可以包含空行（不含任何字符）及註釋行（行首爲 # 符號）。

以製表符（Tab）分隔的第一個字段是所定義的文字，可以是單字或詞組；

第二個字段是與文字對應的編碼；若該編碼由多個「音節」組成，音節之間以空格分開；

可選地、第三個字段是設定該字詞權重的頻度值（非負整數），或相對於預設權值的百分比（浮點數%）。 在拼音輸入法中，往往多音字的若干種讀音使用的場合不同，於是指定不同百分比來修正每個讀音的使用頻度。

詞組如果滿足以下條件，則可以省去編碼字段：

- 詞組中每個單字均有編碼定義
- 詞組中不包含多音字（例：你我），或多音字在該詞組中讀音的權值超過該多音字全部讀音權值的5%（例：我的）

這種條件下，詞組的編碼可由單字編碼的組合推導出來。

反之，則有必要給出詞組的編碼以消除自動註音的不確定性（例：天地）。

當含有多音字的詞組缺少編碼字段時，自動註音程序會利用權重百分比高於5%的讀音進行組合、生成全部可能的註音，如：

「好好地」在編譯時自動註音爲「`hao hao de`」、「`hao hao di`」

設定項速查手冊

雪齋的文檔 全面而詳細解釋了輸入方案及詞典中各設定項的含義及用法。

八股文

Rime 有一份名爲【八股文】的預設詞彙表。

多數輸入方案需要用到一些標準白話文中的通用詞彙。爲免重複在每份碼表中包含這些詞彙，減少輸入方案維護成本，Rime 提供了一份預設詞彙表及自動編碼（註音）的設施。

創作輸入方案時，如果希望完全控制詞彙表的內容而不採用【八股文】中的詞組，只須直接將詞彙編入碼表即可。

否則，在詞典文件中設定 `use_preset_vocabulary: true` 將【八股文】導入該詞典； 在碼表中給出單字碼、需要分辨多音字的詞組編碼、以及該輸入方案特有的詞彙，其他交給自動註音來做就好啦。

Rime預設輸入方案正是利用這份詞彙表及自動註音工具，在不犧牲效果及可維護性的前提下、使詞典文件壓縮到最小的行數。

【八股文】包含從 CC-CEDICT、新酷音等開源項目中整理出來的約二十三萬條詞彙，其用字及詞頻數據針對傳統漢字做過調整。因此基於這份詞彙表產生的輸入結果，比較接近以傳統漢字行文的實際用法。

爲了充分利用【八股文】提供的詞彙，自定義的詞典應保證單字碼表收錄了符合 opencc 字形標準的常用字。特別注意，該標準對以下幾組異體字的取捨，【八股文】將這些字（包括詞組及字頻）統一到左邊一列的字形。

```
爲　　為
僞　　偽
媧　　媯
溈　　潙
兇　　凶
啓　　啟
棱　　稜
污　　汙
泄　　洩
涌　　湧
牀　　床
着　　著
竈　　灶
衆　　眾
裏　　裡
踊　　踴
麪　　麵
羣　　群
峯　　峰
```

請務必在碼表中收錄左列的單字；並建議收全右列的單字。

輸入法詞典往往對下列幾組字不做嚴格區分，opencc 則傾向於細分異體字的不同用法。

```
喫　　吃
黴　　霉
攷　　考
覈　　核
```

請儘量在碼表中收全以上單字。

部署過程中，未能完成自動註音的字、詞會以警告形式出現在日誌文件裏。如果所報告的字爲生僻字、您可以忽略他；如果警告中大量出現某個常用字，那麼應該注意到碼表裏缺失了該字的註音。

編譯輸入方案

將寫好的輸入方案佈署到 Rime 輸入法的過程，稱爲「編譯」：

爲查詢效率故，輸入法工作時不直接加載文本格式的詞典源文件，而要在編譯過程中，爲輸入方案生成專爲高速查詢設計的「.bin」文件。

編譯時程序做以下幾件事：

- 將用戶的定製內容合併到輸入方案定義中，在用戶資料夾生成 .schema.yaml 文檔副本；
- 依照輸入方案中指定的詞典：求得音節表（不同種編碼的集合）、單字表；
- 對詞典中未提供編碼的詞組自動註音，也包括從【八股文】導入的詞組；
- 建立按音節編碼檢索詞條的索引，製作 Rime 固態詞典；
- 建立按詞條檢索編碼的索引，製作 Rime 反查詞典；
- 依照音節表和方案定義中指定的拼寫運算規則，製作 Rime 棱鏡。

佈署 Rime

初次安裝 Rime 輸入法，無有任何輸入方案和用戶設定。因此安裝的最後一個步驟即是把發行版預設的輸入方案和設定文件佈署到 Rime 爲該用戶創建的工作目錄，至此 Rime 纔成爲一部可以發動的輸入引擎。

此後無論是修改已有方案的設定，或是添加了新的輸入方案，都需要「重新佈署」成功後方可使用戶的新設定作用於 Rime 輸入法。

〔★〕重新佈署的方法是：

- 【小狼毫】從開始菜單選擇「重新部署」；或當開啓托盤圖標時，在托盤圖標上右鍵選擇「重新佈署」；
- 【鼠鬚管】在系統語言文字選單中選擇「重新佈署」；
- 【中州韻】點擊輸入法狀態欄（或IBus菜單）上的 ⟲ (Deploy) 按鈕；

- 早於 ibus-rime 0.9.2 的版本：刪除用戶資料夾的 `default.yaml` 之後、執行 `ibus-daemon -drx` 重載 IBus

定製指南

Rime 輸入方案，將 Rime 輸入法的設定整理成完善的、可分發的形式。 但並非一定要創作新的輸入方案，才可以改變 Rime 的行爲。

當用戶需要對 Rime 中的各種設定做小幅的調節，最直接、但不完全正確的做法是：編輯用戶資料夾中那些 .yaml 文檔。

這一方法有弊端：

- 當 Rime 軟件升級時，也會升級各種設定檔、預設輸入方案。用戶編輯過的文檔會被覆寫爲更高版本，所做調整也便丟失了。
- 即使在軟件升級後再手動恢復經過編輯的文件，也會因設定檔的其他部分未得到更新而失去本次升級新增和修復的功能。

因此，對於隨 Rime 發行的設定檔及預設輸入方案，推薦的定製方法是：

創建一個文件名的主體部份（「.」之前）與要定製的文件相同、次級擴展名（位於「.yaml」之前）寫作 `.custom` 的定製檔，形如：

```
patch:
  "一級設定項/二級設定項/三級設定項": 新的設定值
  "另一個設定項": 新的設定值
  "再一個設定項": 新的設定值
  "含列表的設定項/@0": 列表第一個元素新的設定值
  "含列表的設定項/@last": 列表最後一個元素新的設定值
  "含列表的設定項/@before 0": 在列表第一個元素之前插入新的設定值（不建議在補靶中使用）
  "含列表的設定項/@after last": 在列表最後一個元素之後插入新的設定值（不建議在補靶中使用）
  "含列表的設定項/@next": 在列表最後一個元素之後插入新的設定值（不建議在補靶中使用）
```

`patch` 定義了一組「補靶」，以源文件中的設定爲底本，寫入新的設定項、或以新的設定值取代舊有的值。

以下這些例子，另載於一篇《定製指南》，其中所介紹的知識和技巧，覆蓋了不少本文未討論的細節，想必對於創作新的輸入方案會有啓發。

- 一例、定製每頁候選數
- 一例、定製標點符號
- 一例、定製簡化字輸出
- 一例、默認英文輸出
- 一例、定製方案選單

重要！創作了新的輸入方案，最後一步就是在「方案選單」裏啓用他。

拼寫運算

應該算是 Rime 輸入法最主要的獨創技術。

概括來說就是將方案中的編碼通過規則映射到一組全新的拼寫形式！ 也就是說能讓 Rime 輸入方案在不修改碼表的情況下、適應不同的輸入習慣。

拼寫運算能用來：

- 改革拼寫法
  - 將編碼映射到基於同一音系的其他拼寫法，如注音、拼音、國語羅馬字相互轉換
  - 重定義注音鍵盤、雙拼方案
- 實現簡拼查詢
- 在音節表上靈活地定義模糊音規則
- 實現音節自動糾錯
- 變換回顯的輸入碼或提示碼，如將輸入碼顯示爲字根、注音符號、帶聲調標註的羅馬字

給力嗎？

★這裏 有介紹拼寫運算的專題文章。

綜合演練

如果你安裝好了Rime卻不會玩，就一步一步跟我學吧。

本系列課程每個步驟的完整代碼可由此查閱：

https://github.com/lotem/rimeime/tree/master/doc/tutorial

【一】破空中出鞘

Hello, Rime!

第一個例子，總是最簡單的（也是最傻的）。

```
# Rime schema
# encoding: utf-8
#
# 最簡單的 Rime 輸入方案
#

schema:
```

```
  schema_id: hello    # 注意此ID與文件名裏 .schema.yaml 之前的部分相同
  name: 大家好        # 將在〔方案選單〕中顯示
  version: "1"        # 這是文字類型而非整數或小數，如 "1.2.3"
```

起首幾行是註釋。而後只有一組必要的方案描述信息。

這一課主要練習建立格式正確的YAML文檔。

- 要點一，讓你的文本編輯器以UTF-8編碼保存該文件；
- 要點二，注意將 `schema:` 之下的三行代碼以空格縮進——我的習慣是用兩個空格——而 **不要** 用Tab字符來縮進。

縮進表示設定項所屬的層次。在他處引用到此文檔中的設定項，可分別以 `schema/schema_id`, `schema/name`, `schema/version` 來指稱。

我現在把寫好的方案文檔命名爲 `hello.schema.yaml` ，丟進用戶資料夾——對，只要這一個文件就妥了；

然後，啓用他。有些版本會有「方案選單設定」這個介面，在那裏勾選【大家好】這個方案即可。若無有設定介面，則按照上文《定製方案選單》一節來做。

好運！我已建立了一款名爲【大家好】的新方案！雖然他沒有實現任何效果、按鍵仍會像無有輸入法一樣直接輸出西文。

開始改裝

爲了處理字符按鍵、生成輸入碼，本例向輸入引擎添加兩個功能組件。

以下代碼仍是ID爲 `hello` 的新款輸入方案，但增加了 `schema/version` 的數值。 以後每個版本，都以前一個版本爲基礎改寫，引文略去無有改動的部分，以突出重點。

```
# ...
schema:
  schema_id: hello
  name: 大家好
  version: "2"

engine:
  processors:
    - fluid_editor
  segmentors:
    - fallback_segmentor
```

`fluid_editor` 將字符按鍵記入輸入上下文， `fallback_segmentor` 將輸入碼連綴成一段。於是重新佈署後，按下字符鍵不再直接上屏，而顯示爲輸入碼。

你會發現，該輸入法只是收集了鍵盤上的可打印字符，並於按下空格、回車鍵時令輸入碼上屏。

現在就好似寫輸入法程序的過程中，將將取得一小點成果，還有很多邏輯未實現。不同的是，在Rime輸入方案裏寫一行代碼，頂 Rime 開發者所寫的上百上千行。因此我可以很快地組合各種邏輯組件、搭建出心裏想的輸入法。

創建候選項

第二版的【大家好】將鍵盤上所有字符都記入輸入碼，這對整句輸入有用，但是時下流行輸入法只處理編碼字符、其他字符直接上屏的形式。爲了對編碼字符做出區分，以下改用 `speller` + `express_editor` 的組合取代 `fluid_editor`：

```
# ...

schema:
  # ...
  version: "3"

engine:
  processors:
    - speller          # 把字母追加到編碼串
    - express_editor   # 空格確認當前輸入、其他字符直接上屏
  segmentors:
    - fallback_segmentor
```

`speller` 默認只接受小寫拉丁字母作爲輸入碼。 試試看，輸入其他字符如大寫字母、數字、標點，都會直接上屏。並且如果已經輸入了編碼時，下一個直接上屏的字符會將輸入碼頂上屏。

再接着，創建一個最簡單的候選項——把編碼串本身作爲一個選項。故而會提供這個選項的新組件名叫 `echo_translator`。

```
# ...

engine:
  # ...
  translators:
    - echo_translator  # （無有其他結果時，）創建一個與編碼串一個模樣的候選項
```

至此，【大家好】看上去與一個真正的輸入法形似啦。只是還不會打出「大家好」哇？

編寫詞典

那就寫一部詞典，碼表中設定以 `hello` 作爲短語「大家好」的編碼：

```
# Rime dictionary
# encoding: utf-8

---
name: hello
```

```
version: "1"
sort: original
...

大家好   hello
再見     bye
再會     bye
```

※注意： **不要** 從網頁複製以上代碼到實作的詞典文件！因爲網頁裏製表符被轉換成空格從而不符合 Rime 詞典要求的格式。

同時修改方案定義：

```
#...

schema:
  # ...
  version: "4"

engine:
  #...
  segmentors:
    - abc_segmentor       # 標記輸入碼的類型
    - fallback_segmentor
  translators:
    - echo_translator
    - table_translator    # 碼表式轉換

translator:
  dictionary: hello       # 設定 table_translator 使用的詞典名
```

工作流程是這樣的：

- `speller` 將字母鍵加入輸入碼序列
- `abc_segmentor` 給輸入碼打上標籤 `abc`
- `table_translator` 把帶有 `abc` 籤的輸入碼以查表的方式譯爲中文
- `table_translator` 所查的碼表在 `translator/dictionary` 所指定的詞典裏

現在可以敲 `hello` 而打出「大家好」。完工！

實現選字及換頁

等一下。

記得 `hello` 詞典裏，還有個編碼叫做 `bye`。敲 `bye`，Rime 給出「再見」、「再會」兩個候選短語。

這時敲空格鍵，就會打出「再見」；那麼怎樣打出「再會」呢？

大家首先想到的方法，是：打完編碼 `bye`，按 `1` 選「再見」，按 `2` 選「再會」。 可是現在按下 `2` 去，卻是上屏「再見」和數字「2」。可見並沒有完成數字鍵選字的處理，而是將數字同其他符號一樣做了頂字上屏處理。

增加一部 `selector`，即可實現以數字鍵選字。

```
schema:
  # ...
  version: "5"

engine:
  processors:
    - speller
    - selector          # 選字、換頁
    - navigator         # 移動插入點
    - express_editor
  # ...
```

`selector` 除了數字鍵，還響應前次頁、上下方向鍵。因此選擇第二候選「再會」，既可以按數字 `2`，又可以按方向鍵「↓」將「再會」高亮、再按空格鍵確認。

`navigator` 處理左右方向鍵、 `Home` 、 `End` 鍵，實現移動插入點的編輯功能。有兩種情況需要用到他：一是發現輸入碼有誤需要定位修改，二是縮小候選詞對應的輸入碼的範圍、精準地編輯新詞組。

接下來向詞典添加一組重碼，以檢驗換頁的效果：

```
---
name: hello
version: "2"
sort: original
...

大家好   hello
再見     bye
再會     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
```

默認每頁候選數爲5，輸入 `weekday` ，顯示「星期一」至「星期五」。再敲 `Page_Down` 顯示第二頁後選詞「星期六、星期日」。

輸出中文標點

```
schema:
  # ...
  version: "6"

engine:
  processors:
    - speller
    - punctuator        # 處理符號按鍵
    - selector
    - navigator
    - express_editor
  segmentors:
    - abc_segmentor
    - punct_segmentor   # 劃界，與前後方的其他編碼區分開
    - fallback_segmentor
  translators:
    - echo_translator
    - punct_translator  # 轉換
    - table_translator

# ...

punctuator:             # 設定符號表，這裏直接導入預設的
  import_preset: default
```

這次的修改，要注意 `punctuator`, `punct_segmentor`, `punct_translator` 相對於其他組件的位置。

`punctuator/import_preset` 告訴 Rime 使用一套預設的符號表。他的值 `default` 可以換成其他名字如 `xxx` ，則 Rime 會讀取 `xxx.yaml` 裏面定義的符號表。

如今再敲 `hello.` 就會得到「大家好。」

用符號鍵換頁

早先流行用 `-` 和 `=` 這一對符號換頁，如今流行用 `,` 和 `.` 。 在第六版中「，」「。」是會頂字上屏的。現在要做些處理以達到一鍵兩用的效果。

Rime 提供了 `key_binder` 組件，他能夠在一定條件下，將指定按鍵綁定爲另一個按鍵。對於本例就是：

- 當展現候選菜單時，句號鍵（ `period` ）綁定爲向後換頁（ `Page_Down` ）

- 當已有（向後）換頁動作時，逗號鍵（`comma`）綁定爲向前換頁（`Page_Up`）

逗號鍵向前換頁的條件之所以比句號鍵嚴格，是爲了「，」仍可在未進行換頁的情況下頂字上屏。

經過 `key_binder` 的處理，用來換頁的逗號、句號鍵改頭換面爲前、後換頁鍵，從而繞過 `punctuator`，最終被 `selector` 當作換頁來處理。

最終的代碼如下：

```
schema:
  schema_id: hello
  name: 大家好
  version: "7"

engine:
  processors:
    - key_binder  # 搶在其他 processor 處理之前判定是否換頁用的符號鍵
    - speller
    - punctuator  # 否則「，。」就會由此上屏
    - 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:             # 每條定義包含條件、接收按鍵（IBus規格的鍵名，可加修飾符，如「Control+Return」）、發送按鍵

    - when:   paging    # 僅當已發生向後換頁時，
      accept: comma     # 將「逗號」鍵……
      send:   Page_Up   # 關聯到「向前換頁」；於是 navigator 將收到一發 Page_Up

    - when:   has_menu  # 只要有候選字即滿足條件
      accept: period
      send:   Page_Down
```

【二】修煉之道

與【大家好】這個方案不同。以下一組示例，主要演示如何活用符號鍵盤，以及羅馬字轉寫式輸入。

改造鍵盤

莫以爲【大家好】是最最簡單的輸入方案。碼表式輸入法，不如「鍵盤式」輸入法來得簡單明快！

用 `punctuator` 這一套組件，就可實現一款鍵盤輸入法：

```
# Rime schema
# encoding: utf-8

schema:
  schema_id: numbers
  name: 數字之道
  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" : 萬
    "n" : 年
    "y" : [ 月, 元, 億 ]
    "r" : 日
    "x" : 星期
    "j" : 角
    "f" : 分
    "z" : [ 之, 整 ]
    "d" : 第
```

```
    "h" : 號
    "." : 點
  full_shape: *symtable
```

對，所謂「鍵盤輸入法」，就是按鍵和字直接對應的輸入方式。

這次，不再寫 `punctuator/import_preset` 這項，而是自訂了一套符號表。

鴿！原來 `punctuator` 不單可以用來打出標點符號；還可以重定義空格以及全部 94 個可打印 ASCII 字符（碼位 0x20 至 0x7e）。

在符號表代碼裏，用對應的ASCII字符表示按鍵。記得這些按鍵字符要放在引號裏面，YAML 纔能夠正確解析喔。

示例代碼表演了兩種符號的映射方式：一對一及一對多。一對多者，按鍵後符號不會立即上屏，而是……嘿嘿，自己體驗吧 `:-)`

關於代碼裏 `symtable` 的一點解釋：

這是YAML的一種語法，`&symtable` 叫做「錨點標籤」，給緊隨其後的內容起個名字叫 `symtable`；`*symtable` 則相當於引用了 `symtable` 所標記的那段內容，從而避免重複。

Rime 裏的符號有「全角」、「半角」兩種狀態。本方案裏暫不作區分，教 `half_shape`、`full_shape` 使用同一份符號表。

大寫數字鍵盤

靈機一動，不如利用「全、半角」模式來區分「大、小寫」中文數字！

```
schema:
  # ...
  version: "2"

switches:
  - name: full_shape
    states: [ 小寫, 大寫 ]

# ...
```

先來定義狀態開關：`0` 態改「半角」爲「小寫」，`1` 態改「全角」爲「大寫」。

這樣一改，再打開「方案選單」，方案「數字之道」底下就會多出個「小寫→大寫」的選項，每選定一次、狀態隨之反轉一次。

接着給 `half_shape`、`full_shape` 定義不同的符號表：

```
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" : 萬
    "n" : 年
    "y" : [ 月, 元, 億 ]
    "r" : 日
    "x" : 星期
    "j" : 角
    "f" : 分
    "z" : [ 之, 整 ]
    "d" : 第
    "h" : 號
    "." : 點
  full_shape:
    "1" : 壹
    "2" : 貳
    "3" : 參
    "4" : 肆
    "5" : 伍
    "6" : 陸
    "7" : 柒
    "8" : 捌
    "9" : 玖
    "0" : 零
    "s" : 拾
    "b" : 佰
    "q" : 仟
    "w" : 萬
    "n" : 年
    "y" : [ 月, 圓, 億 ]
    "r" : 日
    "x" : 星期
    "j" : 角
    "f" : 分
    "z" : [ 之, 整 ]
    "d" : 第
    "h" : 號
    "." : 點
```

哈，調出選單切換一下大小寫，輸出的字全變樣！酷。

但是要去選單切換，總不如按下 `Shift` 就全都有了：

```
punctuator:
  half_shape:
    # ... 添加以下這些
    "!" : 壹
    "@" : 貳
    "#" : 參
    "$" : [ 肆, ￥, "$", "€", "£" ]
    "%" : [ 伍, 百分之 ]
    "^" : 陸
    "&" : 柒
    "*" : 捌
    "(" : 玖
    ")" : 零
    "S" : 拾
    "B" : 佰
    "Q" : 仟
    "Y" : 圓
```

於是在「小寫」態，只要按 `Shift` + 數字鍵即可打出大寫數字。

用了幾下，發現一處小小的不滿意：敲 `$` 這個鍵，可選的符號有五個之多。想要打出畝元、英鎊符號只得多敲幾下 `$` 鍵使想要的符號高亮；但是按上、下方向鍵並沒有效果，按符號前面標示的數字序號，更是不僅上屏了錯誤的符號、還多上屏一個數字——

這反映出兩個問題。一是 `selector` 組件缺席使得選字、移動選字光標的動作未得到響應。立即加上：

```
# ...

engine:
  processors:
    - punctuator
    - selector        # 加在這裏
    - express_editor
  # ...
```

因爲要讓 `punctuator` 來轉換數字鍵，所以 `selector` 得放在他後頭。

好。二一個問題還在：無法用數字序號選字。爲解決這個衝突，改用閒置的字母鍵來選字：

```
# ...

menu:
  alternative_select_keys: "acegi"
```

完工。

羅馬字之道

畢竟，鍵盤上只有47個字符按鍵、94個編碼字符，對付百十個字還管使。可要輸入上千個常用漢字，嫌鍵盤式輸入的編碼空間太小，必得採用多字符編碼。

羅馬字，以拉丁字母的特定排列作爲漢語音節的轉寫形式。一個音節代表一組同音字，再由音節拼寫組合成詞、句。

凡此單字（音節）編碼自然連用而生詞、句的輸入法，皆可用 `script_translator` 組件完成基於音節碼切分的智能詞句轉換。他有個別名 `r10n_translator` —— `r10n` 爲 `romanization` 的簡寫。但不限於「拼音」、「注音」、「雙拼」、「粵拼」等一族基於語音編碼的輸入法：形式相似者，如「速成」，雖以字形爲本，亦可應用。

現在來把【數字之道】改成拼音→中文數字的變換。

```
schema:
  schema_id: numbers
  name: 數字之道
  version: "3"

engine:
  processors:
    - speller
    - punctuator
    - selector
    - express_editor
  segmentors:
    - abc_segmentor
    - punct_segmentor
  translators:
    - punct_translator
    - script_translator

translator:
  dictionary: numbers

punctuator:
  half_shape: &symtable
    "!" : 壹
    "@" : 貳
    "#" : 參
    "$" : [ 肆, ￥, "$", "€", "£" ]
    "%" : [ 伍, 百分之 ]
    "^" : 陸
    "&" : 柒
    "*" : 捌
    "(" : 玖
    ")" : 零
    "S" : 拾
    "B" : 佰
    "Q" : 仟
    "W" : 萬
    "N" : 年
    "Y" : [ 月, 圓, 億 ]
```

```
    "R" : 日
    "X" : 星期
    "J" : 角
    "F" : 分
    "Z" : [ 之, 整 ]
    "D" : 第
    "H" : 號
    "." : 點
  full_shape: *symtable
```

符號表裏，把小寫字母、數字鍵都空出來了。小寫字母用來拼音，數字鍵用來選重。重點是本次用了 `script_translator` 這組件。與 `table_translator` 相似，該組件與 `translator/dictionary` 指名的詞典相關聯。

編製詞典：

```
# 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
萬	wan
億	yi
年	nian
月	yue
日	ri
星	xing
期	qi
時	shi
分	fen
秒	miao
元	yuan
角	jiao
```

```
之	zhi
整	zheng
第	di
號	hao
點	dian
是	shi
```

※注意： **不要** 從網頁複製以上代碼到實作的詞典文件！因爲網頁裏製表符被轉換成空格從而不符合 Rime 詞典要求的格式。

※注意： 常用編輯器如VS editor以及Notepad++等軟體遇見*.yaml檔案時會預設禁止tab(ascii hex code 09)的使用，而將使用者鍵入的tab按鍵轉換成兩個空白(ascii hex code 20 20)。請一定要使用製表符來進行詞典文件的編輯。

碼表裏給出了一個「示例」規格的小字集。其中包含幾組重碼字。

要訣 `sort: by_weight` 意圖是不以碼表的順序排列重碼字，而是比較字頻。那字頻呢？沒寫出來。

要訣 `use_preset_vocabulary: true` 用在輸入方案需要支持輸入詞組、而碼表中詞組相對匱乏時。編譯輸入方案期間引入 Rime 預設的【八股文】詞彙——及詞頻資料！這就是碼表中未具字頻的原因。

使用【八股文】，要注意碼表所用的字形是否與該詞彙表一致。八股文的詞彙及詞頻統計都遵照 opencc 繁體字形標準。

如果缺少單字的編碼定義，自然也無法導入某些詞彙。所以本方案只會導入這個數字「小字集」上的詞彙。

用拼寫運算定義簡碼

如今有了一款專門輸入數字的拼音輸入法。比一比昇陽拼音、朤月拼音和地球拼音，還有哪裏不一樣？

很快我發現敲 `xingqiwu` 或 `xingqiw` 都可得到來自【八股文】的詞組「星期五」，這很好。可是敲 `xqw` 怎會不中呢？

原來 `script_translator` 羅馬字中譯的方法是，將輸入碼序列切分爲音節表中的拼寫形式，再按音節查詞典。不信你找本詞典瞧瞧，是不是按完整的拼音（注音）編排的。Rime 詞典也一樣。並沒有 `xqw` 這樣的檢索碼。

現在我要用 Rime 獨門絕活「拼寫運算」來定義一種「音序查字法」。令 `x` 作 `xing` 的簡碼，`q` 作數字之道所有音節中起首爲 `q` 者的簡碼，即略代音節 `qi` 與 `qian`。

「漢語拼音」裏還有三個雙字母的聲符，`zh, ch, sh` 也可做簡碼。

添加拼寫運算規則：

```
schema:
  # ...
  version: "4"

#...

speller:
  algebra:
    - 'abbrev/^([a-z]).+$/$1/'
    - 'abbrev/^([zcs]h).+$/$1/'
```

如此 Rime 便知，除了碼表裏那些拼音，還有若干簡碼也是行得通的拼寫形式。再輸入 `xqw`，Rime 將他拆開 `x'q'w`，再默默對應到音節碼 `xing'qi'wan`、`xing'qi'wu`、`xing'qian'wan` 等等，一翻詞典就得到了一個好詞「星期五」，而其他的組合都說不通。

現在有無有悟到，羅馬字轉寫式輸入法與碼表式輸入法理念上的不同？

哈，做中了。試試看 `sss,sss,sssss,sssss`

卻好像不是我要的「四是四，十是十，十四是十四，四十是四十」……

好辦。如果某些詞彙在方案裏很重要，【八股文】又未收錄，那麼，請添加至碼表：

```
---
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。演示完畢。當然休想就此把 Rime 全盤掌握了。一本《指南書》，若能讓讀者入門，我止說「善哉〜」

再往後，就只有多讀代碼，纔能見識到各種新穎、有趣的玩法。

【三】最高武藝

〔警告〕最後這部戲，對智力、技術功底的要求不一般。如果讀不下去，不要怪我、不要懷疑自己的智商！

即使跳過本節書也無妨，只是不可忽略了下文《關於調試》這一節！（重要哇……）

請檢查是否：

- ※ 已將前兩組實例分析透徹
- ※ 學習完了《拼寫運算》
- ※ 知道雙拼是神碼
- ※ 預習 Rime 預設輸入方案之【朙月拼音】

設計一款【智能ABC雙拼】輸入方案做練習！

```
# Rime schema
# encoding: utf-8

schema:
  schema_id: double_pinyin_abc  # 專有的方案標識
  name: 智能ABC雙拼
  version: "0.9"
  author:
    - 佛振 <chen.sst@gmail.com>
  description: |
    朙月拼音，兼容智能ABC雙拼方案。

switches:
  - name: ascii_mode
    reset: 0
    states: [ 中文, 西文 ]
  - name: full_shape
    states: [ 半角, 全角 ]
  - name: simplification
    states: [ 漢字, 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
    - script_translator
    - reverse_lookup_translator
  filters:
```

```
    - simplifier
    - uniquifier

speller:
  alphabet: zyxwvutsrqponmlkjihgfedcba  # 呃，倒背字母表完全是個人喜好
  delimiter: " '"  # 隔音符號用「'」；第一位的空白用來自動插入到音節邊界處
  algebra:  # 拼寫運算規則，這個纔是實現雙拼方案的重點。寫法有很多種，當然也可以把四百多個音節碼一條一條地列舉
    - erase/^xx$/             # 碼表中有幾個拼音不明的字，編碼成xx了，消滅他
    - derive/^([jqxy])u$/$1v/
    - xform/^zh/A/            # 替換聲母鍵，用大寫以防與原有的字母混淆
    - xform/^ch/E/
    - xform/^sh/V/
    - xform/^([aoe].*)$/O$1/  # 添上固定的零聲母o，先標記爲大寫O
    - xform/ei$/Q/            # 替換韻母鍵
    - xform/ian$/W/           # ※2
    - xform/er$|iu$/R/        # 對應兩種韻母的；音節er現在變爲OR了
    - xform/[iu]ang$/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/           # 檢查一下在此之前是否已轉換過了帶介音的ang；好，※1處有了
    - xform/an$/J/            # 如果※2、※3還無有出現在上文中，應該把他們提到本行之前
    - xform/iao$/Z/           # 對—像這樣讓iao提前出場
    - xform/ao$/K/
    - xform/in$|uai$/C/       # 讓uai提前出場
    - xform/ai$/L/
    - xform/ie$/X/
    - xform/ou$/B/
    - xform/un$/N/
    - xform/[uv]e$|ui$/M/
    - xlit/QWERTYOPASDFGHJKLZXCVBNM/qwertyopasdfghjklzxcvbnm/  # 最後把雙拼碼全部變小寫

translator:
  dictionary: luna_pinyin     # 與【朙月拼音】共用詞典
  prism: double_pinyin_abc    # prism 要以本輸入方案的名稱來命名，以免把朙月拼音的拼寫映射表覆蓋掉
  preedit_format:             # 這段代碼用來將輸入的雙拼碼反轉爲全拼顯示；待見雙拼碼的可以把這段拿掉
    - xform/o(\w)/0$1/        # 零聲母先改爲0，以方便後面的轉換
    - xform/(\w)q/$1ei/       # 雙拼第二碼轉換爲韻母
    - xform/(\w)n/$1un/       # 提前轉換雙拼碼 n 和 g，因爲轉換後的拼音裏就快要出現這兩個字母了，那時將難以分辨出雙拼碼
    - xform/(\w)g/$1eng/      # 當然也可以採取事先將雙拼碼變爲大寫的辦法來與轉換過的拼音做區分，可誰讓我是高手呢
    - xform/(\w)w/$1ian/
    - xform/([dtnljqx])r/$1iu/  # 對應多種韻母的雙拼碼，按搭配的聲母做區分（最好別用排除式如[^o]r 容易出狀況）
    - xform/0r/0er/             # 另一種情況，注意先不消除0，以防後面把e當作聲母轉換爲ch
    - xform/([nljqx])t/$1iang/
    - xform/(\w)t/$1uang/       # 上一行已經把對應到 iang 的雙拼碼 t 消滅，於是這裏不用再列
```

```
舉相配的聲母
    - 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/       # 默默檢查：雙拼碼 o 已經轉換過了
    - 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/"  # 復原聲母，音節開始處的雙拼字母a改寫爲zh；其他位置的才真正是
a
    - "xform/(^|[ '])e/$1ch/"
    - "xform/(^|[ '])v/$1sh/"
    - xform/0(\w)/$1/          # 好了，現在可以把零聲母拿掉啦
    - xform/([nljqxy])v/$1ü/   # 這樣纔是漢語拼音 :-)

reverse_lookup:
  dictionary: cangjie5
  prefix: "`"
  tips: 〔倉頡〕
  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難卜
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]*$"
```

完畢。

這是一道大題。通過改造拼寫法而創作出新的輸入方案。

【四】標準庫

如果需要製作完全屬於自己的輸入方案，少不了要瞭解 Rime 的標準庫。此時，請客倌品讀《Rime方案製作詳解》。更多新意，就在你的筆下！

關於調試

如此複雜的輸入方案，很可能需要反覆調試方可達到想要的結果。

請於試驗時及時查看日誌中是否包含錯誤信息。日誌文件位於：

- 【中州韻】 `/tmp/rime.ibus.*`
- 【小狼毫】 `%TEMP%\rime.weasel.*`
- 【鼠鬚管】 `$TMPDIR/rime.squirrel.*`
- 各發行版的早期版本 `用戶資料夾/rime.log`

按照日誌的級別分爲 INFO / 信息、WARNING / 警告、ERROR / 錯誤。 後兩類應重點關注，如果新方案部署後不可用或輸出與設計不一致，原因可能在此。

沒有任何錯誤信息，就是不好使，有可能是碼表本身的問題，比如把碼表中文字和編碼兩列弄顛倒了——Rime 等你輸入由漢字組成的編碼，然而鍵盤沒有可能做到這一點（否則也不再需要輸入法了）。

後續有計劃爲輸入方案創作者開發名爲「拼寫運算調試器」的工具，能夠較直觀地看到每一步拼寫運算的結果。有助於定義雙拼這樣大量使用拼寫運算的方案。

東風破

「東風破早梅，向暖一枝開。」

構想在 Rime 輸入軟件完善後，能夠連結漢字字形、音韻、輸入法愛好者的共同興趣，形成穩定的使用者社羣，搭建一個分享知識的平臺。

【東風破】，定義爲配置管理器及 Rime 輸入方案倉庫，是廣大 Rime 用家分享配置和輸入方案的平臺。

Rime 是一款強調個性的輸入法。

Rime 不要定義輸入法應當是哪個樣、而要定義輸入法可以玩出哪些花樣。

Rime 不可能通過預設更多的輸入方案來滿足玩家的需求；真正的玩家一定有一般人想不到的高招。

未來一定會有，【東風破】（註：現已投入運行），讓用家輕鬆地找到最新、最酷、最適合自己的 Rime 輸入方案。

▼ Pages 25

Find a Page...

Clone this wiki locally

```
https://github.com/rime/home.wiki.git
```