

目录

目录	1
Home	3
Rime前端汇总	4
Rime经典资料汇总-菜鸟书评	5
致第一次安装 RIME 的你	5
Rime 输入方案设计书	5
定制指南	5
Schema.yaml 详解	5
佛振教你写 Rime 输入方案之辅助码的作法	6
UserGuide	7
简易索引	7
_Sidebar	9
Trimer小知识	9
配置同文输入法	9
案例参考	9
rime-的独门绝技-菜鸟书评	10
Rime的独门绝技	10
拼写运算	10
按键响应的自定义	10
对输入处理流程的建模	10
还缺少一个很重要的功能	10
trime.yaml详解	12
必知必会	12
一、style	12
1. 功能	12
示例：开启英文模式句首自动大写	12
示例：在预设26键键盘上添加语音输入键	12
2. 字体	13
3. 尺寸	13
示例：更改字体	14
示例：局部尺寸微调	14
4. 悬浮窗口	15
示例：自定义悬浮窗	16
5. 其它	17
二、fallback_colors	17
三、preset_color_schemes	17
颜色值	17
配色方案	18
示例：制作一个配色方案	18
示例：局部颜色微调	20
四、android_keys	22
五、preset_keys	23
示例：调整按键属性	23
六、preset_keyboards	25
键盘布局	25
布局调用	25
示例：指定朗月拼音使用36键键盘布局	26
布局调整	26
表1 按键功能组合示例	26

示例：给键盘添加删词功能	27
示例：新建一个副键盘	28
常见问题：修改不生效？	29
附录： schema.yaml中的trime	30
1、style	30
2、switches	30
trimer小知识(1)---Yaml文件开头注释是什么意思？	32
第一行详解	32
第二行详解	32
更多的vim设置	32
trimer小知识(2)---配置文件中的的一些yaml语法	34
yaml小知识	34
Yaml简介	34
数据表示方法	34
数据引用方式	35
列表数据的引用:	35
映射数据的引用:	35
打补丁注意事项	35
其他	36
五笔双键配置案例详解(一)-准备篇	37
五笔双键配置案例详解(一) 准备篇	37
一般的调试流程	37
文本编辑器	37
调试流程	37
定制输入方案	37
五笔双键配置案例详解(三)-用模糊音实现双键转换	39
具体的配置	39
原理说明	39
使用和存在的问题	39
共享码表的问题	39
模糊化后的词频问题	40
修改后的源码	40
五笔双键配置案例详解(二)-添加一个输入方案	42
五笔双键配置案例详解(二) 添加一个输入方案	42
得到输入方案	42
生成新的输入方案	42
添加方案到系统列表	42
提高篇	42
小结	43
五笔双键配置案例详解(四)-实现手机上的双键键盘	44
具体的配置	44
使用说明	45
原理说明	45

Home

Welcome to the trime wiki!

請從側邊欄定位至相關內容 

Rime前端汇总

Frontend	OS	Code	Download
Weasel (小狼毫)	Windows	rime/weasel	link
rime-gits	Windows	lotem/rime-gits	link
PIME	Windows	EasyIME/PIME	link
Squirrel (鼠须管)	Mac OS X	rime/squirrel	link
XIME	Mac OS X	stackia/XIME	link
ibus-rime	Linux	rime/ibus-rime	link
fcitx-rime	Linux	fcitx/fcitx-rime	link
Trime (同文)	Android	osfans/trime	link
iRime	iOS	jimmy54/iRime	App Store

Rime经典资料汇总-菜鸟书评

关于 Rime 的大多数权威资料都在 Rime 的 wiki 上都可以找得到，还有一些贴吧经典的帖子。

全都列在这里，做一个汇总，顺便大家可以加上一些书评和介绍，方便大家查阅。

另外，网上以前的有些链接都失效的，主要是 Google 相关的，都被墙了，大家注意不要点。

- [Rime 官网](#)
- [Rime 贴吧](#)

致第一次安装 RIME 的你

入门，快速上手一篇文章。作者直奔主题，开始讲解各种定制，适合入门学习。

这里给的是贴吧的链接，从帖子里，大家可以看到网友的提问和作者的回答，更有助于理解。

Rime 输入方案设计书

Rime 输入方案创作者的第一本参考书 ※ 佛振 chen.sst@gmail.com 修訂於 2013 年 5 月 4 日

关于 Rime，最最权威，最最经典的资料。出自 Rime 的主要设计者和创作者 佛振老大之手。进阶必读！

对Rime的工作原理和设计思想都进行了深入浅出讲解。主要包括以下几个部分：

1. Rime 的简介和一些设计思想
2. Rime 的数据文件分布及作用 ([强烈推荐阅读](#),打 patch 的基础)
3. 详解了输入法各功能组件的工作流程 (适合输入方案设计师和高阶用户，看完后，你会对 yaml 文件中顶层的标签不再陌生)
4. 介绍了如何打 patch 定制 ([强烈推荐阅读](#)，所有定制的基础)
5. 综合演练 (介绍了几个实例，关于输入方案定制，进阶用户必读，想要对现有方案做大改动的必读)

定制指南

这里全是一些具体的细节的定制实例。

如：

- / 鍵輸入標點「、」
- 全套西文标点符号
- 默認英文輸出
- Control 鍵切换中西文 等等

Rime 输入方案设计书 的第5部分，讲的是演练实例，是大的输入方案的改造，而这里只是一些我们日常生活是经常会用到的，对输入方案的小小的调整。

如果你有什么关于定制的问题想要提问，建议先认真阅读一下这里。

当然，如果你解决了某个定制问题，你也可以将解决方案添加到这个 wiki 中。

Schema.yaml 詳解

对一个输入方案的yaml文档做了详解，对 Rime输入方案设计书 中的第三部分的补充，设计书是从理论角度讲，这里是结合schema.yaml文件逐步讲的。

另外文章末尾还附了 Dict.yaml 詳解，如果想对词库或者码表作修改，可以看这里。

下面还有一些比较好的贴子。

佛振教你写 Rime 输入方案之辅助码的作法

可以看作者如何操作 Vim 和正则表达式。对码表进行深入的剖解。


UserGuide

简易索引

- 下载及安装
 - 稳定版：[酷安](#)，[Google Play](#)，[Releases](#)
 - 测试版：目前主要发布在 [Rime同文堂](#) (480159874)
- 添加输入方案
 - [简易图解](#) (待更新)
 - [常用输入方案](#) (或至 [Rime同文堂](#) 下载“Brise懒人包”)
 - [更多开源输入方案](#)
- 学习如何DIY
 - 配置输入法 (键盘及各种界面功能)，可参考 [trime.yaml](#) 详解。
 - 配置输入方案，请参阅 [Rime说明书](#)。

附表1: (同文) Rime 文件分布、作用及相关教程

文件&文件夹	作用及相关教程
 <词典名>.userdb	用户词典：存储用户输入习惯。
 backgrounds	主题图片(可选)：主题中要用到的图片都是放在这里。
 build	编译结果：部署成功后，会在此处生成编译结果文件（yaml 或 bin 格式）。输入法程序运行时读取的也是这里的文件。对于较复杂的输入方案，在手机端若无法部署，也可将 PC 端部署生成的编译结果文件拷贝到这里使用（新版 librime 生成的 bin 文件可通用）。编辑方案或主题请直接操作用户文件夹中的源文件，而非这里的编译结果。
 fonts	自定义字体(可选)：用于改变界面字体。将个性化的字体存放于此文件夹中，再在 trime.yaml 中调用，示例： 更改界面字体
 openccc	简繁转换组件(可选)：简繁转换。 原理及示例
 sync	同步文件夹：备份方案&词库及相关配置文件，导出的用户词典也存放在此处。详见 同步用户资料 。
 custom_phrase.txt	自定义短语(可选)：存储少量的固定短语等数据。配置步骤：① 新建短语翻译器 ② 配置翻译器 ③往custom_phrase.txt添加自定义短语（ custom_phrase样例文件 *）
 default.yaml  default_custom.yaml	全局设定及其补丁文件：Rime各个平台通用的全局参数 (功能键定义、按键捆绑、方案列表、候选条数……)。请参考 定制指南
 essay.txt	八股文(可选)：一份词汇表和简陋的语言模型。 八股文的详细说明
 installation.yaml	安装信息：保存安装ID用以区分不同来源的备份数据，也可以在此处设定同步位置。详见 同步用户资料
 <方案标识>.schema.yaml  <方案标识>.custom.yaml	输入方案定义及其补丁文件：输入方案的设定。可参考 详解输入方案 以及 schema.yaml 详解
 <词典名>.dict.yaml  <词典名>.<分词库名>.dict.yaml	输入方案词典及其分词库：输入方案所使用的词典(包含词条、编码、构词码、权重等信息)。详见 码表与词典 以及 dict.yaml 详解
 symbols.yaml	扩充的特殊符号：提供了比 default.yaml 更为丰富的特殊符号， symbols.yaml用法说明 。
 trime.yaml  trime_custom.yaml  xxx.trime.yaml  xxx.trime_custom.yaml	同文主题及其补丁文件：定义键盘配色、布局、样式等。可参考 trime.yaml 详解

 user.yam	用户状态信息：用来保存当前所使用的方案ID，以及各种开关的状态。

_Sidebar

Trimer小知识

- [Rime前端汇总](#)
- [Rime经典资料汇总](#)
- [Yaml文件开头注释是什么意思？](#)
- [配置文件中的某些yaml语法](#)

配置同文输入法

- [简易索引](#)
- [前端配置](#) ([trime.yaml](#)詳解)

案例参考

- [五笔双键配置案例详解\(待完善\)](#)
 - [\(一\) 准备篇](#)
 - [\(二\) 添加一个输入方案](#)
 - [\(三\) 用模糊音实现双键转换](#)
 - [\(四\) 实现手机上的双键键盘](#)

rime-的独门绝技-菜鸟书评

Rime的独门绝技

还没有回复佛振大大的问题。 Rime的独门绝技主要有：

拼写运算

利用正则表达式，重新定义了一套简单的规则，很好的解决了码表变换的问题。 在利用现有码表，编辑生成新的输入法方案时，特别有效。

这个方案的优点是：

1. 特别简洁有效，抓住了问题的核心，对输入模型
2. 实现起来性价比极高，可以转换为标准的正则表达式
3. 扩展性良好，正则表达式的强大和功能完善无需质疑

按键响应的自定义

1. 可以为键盘的每个按键指定事件响应
 - 主要用在标点符号的中英输出精确控制，程序员可以在中文状态下输出某些中文生僻字符的英文标点
 - 按键选2, 3候选码
2. 可以自定义按键
 - 我在rime里面用过，自定义一个按键，切换键盘
 - 自定义一个键，用来输入日期+时间

对输入处理流程的建模

建立的模型很好，利用了Unix一切都是字符流的思想，还有kiss原则。

从按键输入流，到最后输出汉字，主要用了四大组件集：

- processors：按键处理
- segmentors：字符分段
- translators：对字符编码转换为汉字
- filters：过滤输出

每件组件集里面，又有一些子组件依次对字符流进行过滤，跟Unix的管道很像。

不过这个我也理解不深，只是试着做一个隐喻：

古代有个由一家四口开的一个药店。

- processors: 父亲是柜台老板，负责与买药的人交流，记下客户的描述和要求；客户只是通过描述药物的信息来说明药物。有时候，还会描述错，又要叫老板把一些描述删掉。
- segmentors: 母亲在柜台旁边协助父亲，帮助整理父亲记下来的客户描述，客户描述得比较多，母亲的主要工作就是把这些描述进行分段，确定哪一段是一种药物。
- translators: 小妹根据母亲整理出来的条子，进行抓药。由于客户描述的药物符合条件的不止一种，小妹会把所有可能的药物都抓好，最后让客户进行挑选。
- filters: 大哥是负责最后对小妹抓出来的药进行一个重新打包（虽然现在都用塑料袋了，有些人还是喜欢古朴的纸包装，就是繁体字了），并且会过滤掉小妹抓重复的药。

大哥把最后的药交给父亲后，父亲又会继续与买药的客户交流，确定最后客户要拿哪个药。

这是一个循环的过程。

还缺少一个很重要的功能

输入字符串也进行正则模糊化的功能。

现在，对码表可以利用正则表达式进行离线处理。 但是，对于输入的字符串也应该包含一个正则表达式的模糊输入的功能。

主要应用场合：

- 手机上的九宫格

- 五笔的Z键作万能键，可能用来代替所有的按键，用来查询编码

trime.yaml詳解

(基于同文V3.01 20180404版修订)

必知必会

您可能需要先了解 YAML 的基本语法。这篇定制指南里有一些实例可以帮助您理解 Rime 的配置方式。另外 Rime 在 YAML 语法的基础上新增了编译指令，想要更灵活地制作同文主题的同学可以参考一下。

一、style

界面样式及特色功能

1. 功能

- `auto_caps` : 自动句首大写 (`true` :打开; `false` :关闭; `ascii` :仅英文模式句首大写)
- `candidate_use_cursor` : 候选焦点高亮 (`true` :打开; `false` :关闭)
- `comment_on_top` : 候选项注释在上方或右侧 (`true` :在上方; `false` :在右侧)
- `horizontal` : 水平模式。改变方向键的功能 (`true` :方向键适配横排候选; `false` :方向键适配竖排候选)
- `keyboards` : 键盘配置。除主键盘外, 其它需要用到的键盘都要在这里声明。
- `proximity_correction` : 将按键之间的空白区域分配给相邻的按键, 避免空按 (`true` :打开; `false` :关闭)
- `reset_ascii_mode` : 不同进程中显示键盘时重置为中文状态 (`true` :重置为中文; `false` :记忆中英状态)
- `latin_locale` : 在英文状态 (`ascii_mode`) 下, 朗读按键时所用的语言。
- `locale` : 在中文状态下, 朗读上屏文本和按键时所用的语言。
※ 需要先在同文设置界面开启朗读功能。朗读功能还需要手机的TTS引擎支持。可使用系统默认引擎, 也可安装讯飞语记等第三方引擎。 `latin_local` 和 `local` 可以设置的语言也取决于 TTS 引擎。常见的语言: `zh_TW`, `zh_CN`, `zh_HK`, `en_US`, `ja_JP`, `ko_KR`, ……
- `speech_opencore_config` : 语音输入简繁转换 (默认值 `s2t.json` : 将语音识别的结果转换成繁体再上屏)
需要配合OpenCC组件来使用。转换的选项有:
 - `s2t.json` #简体→繁体
 - `t2s.json` #繁体→简体
 - `s2hk.json` #简体→香港繁体
 - `hk2s.json` #香港繁体→简体
 - `s2tw.json` #简体→台湾繁体
 - `s2twp.json` #简体→台湾繁体, 并转换常用词汇 (网络→網路)
 - `tw2s.json` #台湾繁体→简体
 - `tw2sp.json` #台湾繁体→简体, 并转换常用词汇 (作業系統→操作系统)
 - `t2hk.json` #OpenCC标准繁体→香港繁体
 - `t2tw.json` #OpenCC标准繁体→台湾繁体
 - 如果不需要转换, 想让语音引擎按原样输出, 可设为 `none`。(2017-9-13开始, 也可以直接注释掉)
△ 同文输入法的语音输入依赖的是手机的「语音识别服务」, 而且必须安装「讯飞语记」或者「讯飞语音+」才能使用。

示例：开启英文模式句首自动大写

```
# trime.custom.yaml
patch:
  "style/auto_caps": ascii
```

示例：在预设26键键盘上添加语音输入键

```
# trime.custom.yaml
patch:
  "preset_keyboards/qwerty/keys/@31/click": VOICE_ASSIST #将原来的符号键替换为语音键
```

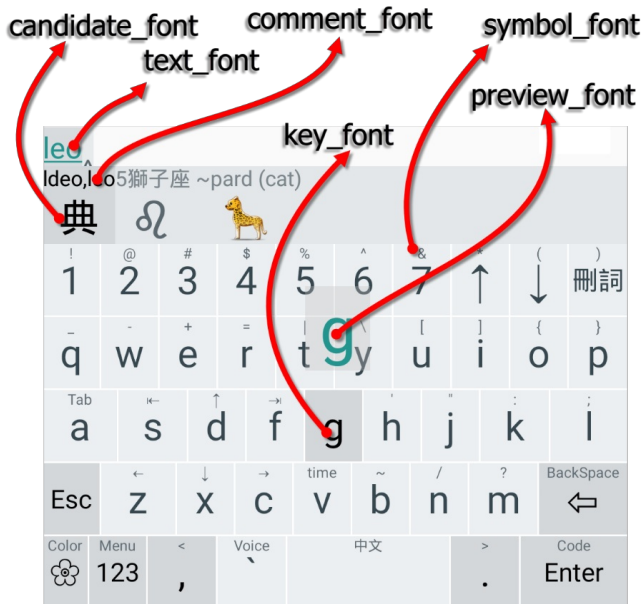
※ 对于默认主题 `trime.yaml`, 修改的时候需要打补丁。对于自制的主题, 一般不需要打补丁。

上例若直接修改主题文件, 是这样做的:

查找按键布局 `qwerty`,

将按键 `{click: Keyboard_symbols, long_click: Keyboard_number}` 修改成 `{click: VOICE_ASSIST, long_click: Keyboard_number}`

2. 字体



- `text_font`: 编码字体
- `label_font`: 悬浮窗候选项序号字体
- `candidate_font`: 候选字体
- `comment_font`: 候选注释字体
- `hanb_font`: 后备字体。用于补充候选字体 (`candidate_font`)。
 - ※ 某些特殊符号, 或者很多生僻字 (比如Unicode Ext-B~Ext-F的字符) 在大多数手机上通常都会显示成方框或空白。`hanb_font` 可以使这些字符在同文输入法里正常显示 (推荐使用花园明朝B字体: HanaMinB.ttf)。您也可以直接在系统中设置fallback字体 (全局生效, 但需要root)。
- `latin_font`: 候选及候选注释拉丁字体 (暂时对悬浮窗候选无效)
 - ※ 当 `latin_font` 生效时, 拉丁字符 (< 0x2e80) 就不再由 `comment_font` 和 `candidate_font` 控制
- `key_font`: 按键字体 (click)
- `symbol_font`: 符号字体 (long_click和hint)
- `preview_font`: 按键气泡字体

3. 尺寸

- `text_size`: 编码大小
- `label_text_size`: 悬浮窗候选项序号大小
- `candidate_padding`: 候选项内边距 (影响候选项的间距)
- `candidate_spacing`: 候选分割线宽度
- `candidate_text_size`: 候选字大小
- `candidate_view_height`: 候选区高度
- `comment_text_size`: 候选注释大小
- `comment_height`: 候选注释区高度
- `key_height`: 键高*
- `key_width`: 键宽*, 占屏幕宽的百分比
 - △ 当按键布局中的 `height` 与 `width` 省略不写时, 此处设置的 `key_height` 与 `key_width` 才会生效。
- `key_text_size`: 按键文本大小 (click)
- `key_long_text_size`: 按键长文本大小 (字数≥2)
- `symbol_text_size`: 符号大小 (long_click和hint)
- `round_corner`: 按键圆角半径
- `preview_height`: 按键气泡高度
- `preview_offset`: 按键气泡纵向偏移 (默认值 -12, 向下偏移为正, 向上偏移为负)
- `preview_text_size`: 按键气泡字体大小
- `shadow_radius`: 键盘字体阴影大小 (数值不宜过大, 可能会造成卡顿)

- horizontal_gap: 键水平间距
- vertical_gap: 键盘行距
 - △ 若关闭了 proximity_correction, 过大的 horizontal_gap 与 vertical_gap 会引起空按漏按
- vertical_correction: 触摸位置校正 (竖直方向)。
 - ※ 为了提升打字手感, 可将按键的实际触摸位置相对其显示位置上下偏移一点点 (默认值 -10, 上偏为正, 下偏为负, 为 0 则不偏移)。

示例：更改字体

①在rime文件夹内新建fonts文件夹

△ fonts文件夹建在共享文件夹与用户文件夹皆可 (若共享文件夹存在fonts, 则字体放在用户文件夹内无效)

②将字体文件复制到fonts文件夹

本例用到了两个字体文件：

- 📄 gunplay.ttf
- 📄 方正行楷简体.ttf

③配置字体参数:

```
# trime.custom.yaml
patch:
  "style/candidate_font": 方正行楷简体.ttf #候选字体
  "style/key_font": 方正行楷简体.ttf #按键字体
  "style/text_font": gunplay.ttf #编码字体
  "style/comment_font": gunplay.ttf #候选注释字体
  "style/symbol_font": gunplay.ttf #符号字体
  "style/candidate_text_size": 28 #候选字体大小
  "style/candidate_view_height": 32 #候选区高度
  "style/comment_height": 16 #候选注释区高度
  "style/comment_text_size": 13 #候选注释字体大小
  "style/key_text_size": 24 #按键字体大小
  "style/round_corner": 0.0 #按键圆角大小
  "style/symbol_text_size": 9 #符号字体大小
  # "style/text_height": 24 #编码区高度 (新版已经取消此参数)
  "style/text_size": 18 #编码字体大小
```

效果图：



示例：局部尺寸微调

style 里的尺寸是全局生效的。实际上我们也可以对某些局部的尺寸做微调。

可以在键盘布局里微调的尺寸：

- horizontal_gap: 键水平间距
- vertical_gap: 键盘行距

- `round_corner` : 按键圆角 (对整个键盘生效)

可以在按键里微调的尺寸：

- `key_text_size` : 按键文本 (对长标签也生效, 不区分按键文本的长短)
- `symbol_text_size` : 符号 (`long_click`和`hint`)
- `round_corner` : 按键圆角 (对单个按键生效)

另外, 按键字符的偏移量也是可以局部微调的, 详见后面 `preset_keyboards`。

例1：调整预设26键键盘布局的水平间距和圆角

```
# trime.custom.yaml
patch:
  "preset_keyboards/qwerty/horizontal_gap": 0 #水平间距改为0
  "preset_keyboards/qwerty/round_corner": 0 #按键圆角改为0
  #以上更改仅对布局ID为qwerty的26键键盘生效
```

例2：单独修改预设26键键盘布局中的空格键

```
# trime.custom.yaml
patch:
  "preset_keyboards/qwerty/keys/@33/key_text_size": 12 #空格键字体改小
  "preset_keyboards/qwerty/keys/@33/round_corner": 32 #圆角增大
```

※上例若直接修改主题文件, 是这样写的：

```
{click: space, key_text_size: 12, round_corner: 32, width: 30}
```

4. 悬浮窗口

- `layout` : 悬浮窗口设置
 - `position` : 悬浮窗位置
 - `left` | `right` | `left_up` | `right_up` 这几种都可以让悬浮窗口动态跟随光标 (需要 \geq Android5.0)
 - `fixed` | `bottom_left` | `bottom_right` | `top_left` | `top_right` 这几种是固定在屏幕的边角上
 - `min_length` : 悬浮窗最小词长 (候选词长大于等于 `min_length` 才会进入悬浮窗)
 - `max_length` : 连续排列的多个候选项总字数 (包括候选项注释) 超过 `max_length` 时, 把超出的候选项移到下一行显示 (单个候选项若超长, 或者 `max_length` 数值过大, 则由 `max_width` 决定是否换行)
 - `sticky_lines` : 固顶行数 (不与其它候选同排, 单独一行显示的候选项个数)
 - `max_entries` : 最大词条数 (允许进入悬浮窗的最大词条数)
 - `all_phrases` : 显示所有长词。所有满足 `min_length` 的词条都显示在悬浮窗 (一般只用于table translator, 有可能会改变候选项的显示顺序)
 - `border` : 边框宽度 (增大边框则向内加粗, 也会对悬浮窗圆角产生一点影响)
 - `max_width` : 窗口最大宽度 (候选超长则自动换行)
 - `min_width` : 最小宽度 (悬浮窗的初始宽度)
 - `margin_x` : 水平边距 (左右留白大小)
 - `margin_y` : 竖直边距 (上下留白大小)
 - `line_spacing` : 候选词的行间距 (px)
 - `line_spacing_multiplier` : 候选词的行间距倍数
 - `spacing` : 悬浮窗位置上下偏移量 (一般上移为正, 下移为负, 但当 `position` 设为`top_xxx`时, 方向是相反的)
 - `round_corner` : 窗口圆角 (同时也会使候选栏的高亮候选边框产生圆角)
 - `alpha` : 悬浮窗透明度* (0x00~0xff. 0x00为全透明)
 - `background` : 悬浮窗背景* (颜色或图片二选一。比如颜色: 0xFFD3FF83; 图片: xxx.jpg。图片格式jpg与png皆可, 相应的图片需放置在使用者文件夹的backgrounds目录下, 放在共享文件夹无效)
 - ※ 当 `background` 设为颜色值时, `alpha` 与 `background` 的透明度是叠加的
 - `elevation` : 悬浮窗阴影 (\geq Android5.0)
 - `movable` : 是否可移动窗口, 或仅移动一次 `true`|`false`|`once`
- `window` : #悬浮窗口组件
 - - {start: "", move: '⬅', end: ""}
 - #窗口移动图标。当 `movable` 设为可移动时, 拖动这个图标即可调整悬浮窗的位置。 `move` 可改为任意符号, `start` `end` 为左右修饰符号, 若不需要修饰可简化为 {move: '⬅'}。
 - - {start: "", composition: "%s", end: "", letter_spacing: 0}

#这个组件用来显示输入的编码。composition 若去掉则不显示编码区。letter_spacing 为字符间距，需要≥Android5.0。

- {start: "\n", label: "%s.", candidate: "%s", comment: " %s", end: "", sep: ""}
#这个组件用来显示候选项。start: "\n" 表示这个组件另起一行，label 候选项序号，candidate 候选项，comment 候选项注释，sep 候选项分隔符。（除 candidate 外，其它都是可选的。比如删掉 label 则不显示候选项序号）
- #~~~~~
※ 另外还可以在悬浮窗内放置普通按键，比如地球拼音的声调键：
 - {start: "\n", click: ";", label: " `", align: center, end: ""}
click: ";" label: "`" 作用与键盘按键相同。align 对齐方式，left左对齐|right右对齐|center居中，默认为左对齐可省略不写（align每行组件只需写一个，也可用于上面的编码与候选）
 - {click: "/", label: " ´", end: ""}
end: "`" 的作用是在按键间形成间隙
 - {click: ",", label: " ˇ", end: ""}
 - {click: "\\", label: " `", end: ""}

示例：自定义悬浮窗

配置几种在平板电脑上的悬浮窗样式：

- 1、横排

```
# trime.custom.yaml
patch:
  "style/horizontal": true
  "style/layout/position": left
  "style/layout/min_length": 1
  "style/layout/max_length": 36
  "style/layout/max_entries": 5
  "style/layout/sticky_lines": 0
  "style/layout/max_width": 930
  "style/layout/margin_x": 0
  "style/layout/margin_y": 0
  "style/layout/border": 0
  "style/layout/round_corner": 3
  "style/layout/elevation": 8
  "style/layout/alpha": 0xff
  "style/layout/line_spacing_multiplier": 1
  "style/window":
    - {label: " %s.", candidate: "%s "}
```

这是同文的hg pl hb xr

1. 横排候选 2. 横排 3. 横拍 4. 横 5. 衡

- 2、竖排

把上面补丁的 sticky_lines 改成 5，水平模式 horizontal 改成 false。

uu pl de yh zi

1. 竖排的样子
2. 竖排
3. 输牌
4. 书
5. 书

- 3、横竖混排

只需要把上面补丁的 sticky_lines 改成 1 即可。

hg uu hp pl

1. 横竖混排
2. 横竖 3. 横 4. 恒 5. 衡

※ 以上示例中去掉了悬浮窗的 `composition` 组件，因此需要开启嵌入模式才能在文本框中显示编码。另外，开启悬浮窗后，也可以把底下多余的候选栏关掉（参考附录中的示例）。

5. 其它

备用参数，暂无功能

- `background_dim_amount`
- `max_height`
- `min_height`

二、 `fallback_colors`

后备颜色：配色方案中未定义的颜色，自动从这里推导。

```
candidate_text_color: text_color
comment_text_color: candidate_text_color
border_color: back_color
candidate_separator_color: border_color
hilited_text_color: text_color
hilited_back_color: back_color
hilited_candidate_text_color: hilited_text_color
hilited_candidate_back_color: hilited_back_color
hilited_comment_text_color: comment_text_color
text_back_color: back_color
hilited_key_back_color: hilited_candidate_back_color
hilited_key_text_color: hilited_candidate_text_color
hilited_key_symbol_color: hilited_comment_text_color
hilited_off_key_back_color: hilited_key_back_color
hilited_on_key_back_color: hilited_key_back_color
hilited_off_key_text_color: hilited_key_text_color
hilited_on_key_text_color: hilited_key_text_color
key_back_color: back_color
key_border_color: border_color
key_text_color: candidate_text_color
key_symbol_color: comment_text_color
keyboard_back_color: border_color
label_color: candidate_text_color
off_key_back_color: key_back_color
off_key_text_color: key_text_color
on_key_back_color: hilited_key_back_color
on_key_text_color: hilited_key_text_color
preview_back_color: key_back_color
preview_text_color: key_text_color
shadow_color: border_color
```

三、 `preset_color_schemes`

预置的配色方案

颜色值

同文支持以下几种写法：

- `0xaarrggbb`
- `"#aarrggbb"`（引号不能省略，否则会跟注释冲突）
- `0xrrggbb`（省略了aa，表示完全不透明）
- `"#rrggbb"`（同上）
- `0xaa`
- `red`、`green`、`blue` ……

其中aa透明度，rr红，gg绿，bb蓝，都是十六进制数值，取值范围00~ff。

配色方案

一个主题中可以有多多个配色方案。

- `default` : 配色方案ID, 不可重复
 - `name` : 配色方案名称
 - `author` : 作者信息
 - ▼悬浮窗口
 - `border_color` : 悬浮窗边框
 - `label_color` : 悬浮窗候选项序号
 - ※ 悬浮窗高亮候选项序号与 `hilited_candidate_text_color` 相同
 - `hilited_text_color` : 高亮编码 (一般是位于光标插入点左边的编码)
 - `text_color` : 编码 (位于光标插入点右边的编码, 或者是拼音类方案中无法正常解析的空码, 比如全拼时输入hau, u就属于这种)
 - `hilited_back_color` : 高亮编码背景
 - ※ 非高亮的编码背景与 `back_color` 相同
 - `text_back_color` : 编码区背景* (编码四周的空白区域, 也是悬浮窗的主背景)
 - ※ 仅当 `style/layout/background` 设置失效时才会起作用 (当 `background` 生效时, `text_back_color` 就会失效)
 - ▼候选项
 - `back_color` : 候选区背景*
 - `hilited_candidate_back_color` : 高亮候选背景 (候选项被选中时)
 - `candidate_separator_color` : 候选分割线
 - `candidate_text_color` : 候选文本 (包括悬浮窗候选, 下同)
 - `hilited_candidate_text_color` : 高亮候选文本
 - `comment_text_color` : 候选项注释
 - `hilited_comment_text_color` : 高亮候选项注释
 - ▼键盘
 - ☆ `key_back_color` : 按键背景
 - ☆ `hilited_key_back_color` : 高亮按键背景 (按下按键时)
 - `key_text_color` : 按键文本 (click)
 - `hilited_key_text_color` : 高亮按键文本
 - `key_symbol_color` : 按键符号 (long_click和hint)
 - `hilited_key_symbol_color` : 高亮按键符号
 - `preview_back_color` : 按键气泡背景
 - `preview_text_color` : 按键气泡文本
 - `shadow_color` : 按键文字阴影 (阴影半径在 `shadow_radius` 中设定)
 - ☆ `keyboard_back_color` : 键盘背景
 - `key_border_color` : 按键边框*(暂无)
 - ▼功能键 (functional: true)
 - ☆ `off_key_back_color` : 功能键背景
 - ☆ `hilited_off_key_back_color` : 功能键高亮背景 (按下时)
 - `off_key_text_color` : 功能键文本
 - `hilited_off_key_text_color` : 功能键高亮文本
 - ※ 在没有特别指定的时候, 功能键的long_click和hint颜色与普通按键一样
 - ☆ `on_key_back_color` : shift键锁定时背景
 - ☆ `hilited_on_key_back_color` : shift键锁定时的高亮背景 (按下时)
 - `on_key_text_color` : shift键锁定时文本
 - `hilited_on_key_text_color` : shift键锁定时的高亮文本
 - ※ shift键锁定时因为这四种颜色不会因为 `functional: false` 而失效
- ※ 以上标记为☆的都可以使用图片作背景 (与悬浮窗背景图做法相同)。

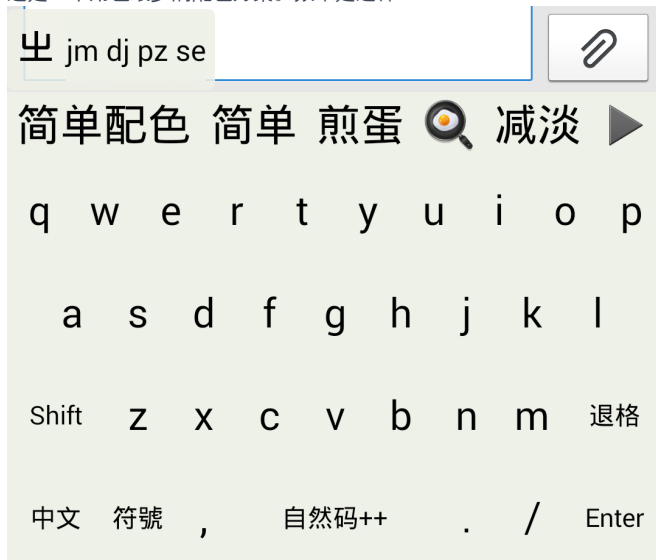
示例：制作一个配色方案

有了 `fallback_colors`, 最少只需要 `back_color` 和 `text_color` 就可以做出一个配色方案。

```
# trime.custom.yaml
patch:
  "preset_color_schemes/xxx": #配色方案ID
```

```
name: xxx极简      #配色名称
back_color: 0xEEF1E7 #背景
text_color: 0x000000 #文字
```

这是一个用色最少的配色方案。效果是这样：



※ 部署完成后，需要从配色菜单中选取刚才添加的配色方案「xxx极简」，才能看到效果。

试试再加两个颜色：

```
# trime.custom.yaml
patch:
  "preset_color_schemes/xxx": #配色方案ID
    name: xxx极简      #配色名称
    back_color: 0xEEF1E7 #背景
    text_color: 0x000000 #文字
    key_back_color: 0xDEEDB1 #按键背景
    hilited_candidate_back_color: 0xD4ED89 #候选高亮背景
```

好像变得更难看了☹：



再加个背景图看看：

```
# trime.custom.yaml
patch:
  "preset_color_schemes/xxx": #配色方案ID
```

```
name: xxx极简      #配色名称
back_color: 0xEEF1E7 #背景
text_color: 0x000000 #文字
key_back_color: 0x60DEEDB1 #按键背景加了透明度，不然会挡住图片
hilited_candidate_back_color: 0x80D4ED89 #候选高亮背景，这个也加了透明度，使颜色减淡一些
keyboard_back_color: xxx.jpg #图片需放在rime/backgrounds文件夹内
```

最后变成这样：



.....

在每个按键上加图片背景会怎样？您若感兴趣可以试试。

△ 图片不需要太大，上例用到的背景图只有32KB。

示例：局部颜色微调

preset_color_schemes 里的颜色是全局生效的。同文也提供了一些方法可以对某些局部的颜色做微调。

可以在键盘布局里微调的颜色：

- ☆ keyboard_back_color：键盘背景

可以在按键里微调的颜色：

- ☆ key_back_color：按键背景（对功能键也有效，下同）
 - ☆ hilited_key_back_color：高亮按键背景（按下按键时）
 - key_text_color：按键文本（click）
 - hilited_key_text_color：高亮按键文本
 - key_symbol_color：按键符号（long_click和hint）
 - hilited_key_symbol_color：高亮按键符号
- ※ 除了颜色值和图片，在按键里还可以使用分组颜色标签，详见下面例2、例3。
※ 若在键盘里调整功能键颜色，则不区分是否锁定。

例1：修改预设26键键盘回车键的颜色

```
# trime.custom.yaml
patch:
  "preset_keyboards/qwerty/keys/@36/key_back_color": 0xFFAE00
```

※ 上例中，我们给预设26键键盘的回车键分配了一个颜色值。这样的话，不管您切换到哪个配色方案，回车键的颜色都固定是 0xFFAE00。若您想更灵活地更改一个按键的颜色，就需要用到分组颜色标签。

例2：使用分组标签定义按键颜色

先来看一个简单的例子：

```
# trime.custom.yaml
patch:
```

```
"preset_keyboards/qwerty/keys/@15/key_back_color": off_key_back_color
"preset_keyboards/qwerty/keys/@15/hilited_key_back_color": hilited_off_key_back_color
```

off_key_back_color 和 hilited_off_key_back_color 是同文默认的功能键颜色标签。在这个补丁中，我们给 qwerty 键盘的 g 键添加了功能键的颜色标签。这样不管切换到什么配色方案，g 键的颜色总会跟功能键保持一致。

除了使用默认的标签，我们还可以定义自己的颜色标签。

例3：自定义分组标签

现在我们要改变数字键盘的数字键颜色，以便快速地与普通按键作区分。

```
# trime.custom.yaml
patch:
#步骤一，在数字键盘的数字键中添加分组颜色标签
"preset_keyboards/number":
  name: 预设数字
  author: "osfans <waxaca@163.com>"
  width: 20
  height: 44
  keys:
    - {click: '+'}
    - {click: '1', key_back_color: k_n_b, hilited_key_back_color: h_k_n_b}
    #k_n_b是自定义的标签名，你可以理解成是key_num_back_color的缩写...
    - {click: '2', key_back_color: k_n_b, hilited_key_back_color: h_k_n_b}
    - {click: '3', key_back_color: k_n_b, hilited_key_back_color: h_k_n_b}
    - {click: '#'}
    - {click: '.'}
    - {click: '4', key_back_color: k_n_b, hilited_key_back_color: h_k_n_b}
    - {click: '5', key_back_color: k_n_b, hilited_key_back_color: h_k_n_b}
    - {click: '6', key_back_color: k_n_b, hilited_key_back_color: h_k_n_b}
    - {click: '%'}
    - {click: '*'}
    - {click: '7', key_back_color: k_n_b, hilited_key_back_color: h_k_n_b}
    - {click: '8', key_back_color: k_n_b, hilited_key_back_color: h_k_n_b}
    - {click: '9', key_back_color: k_n_b, hilited_key_back_color: h_k_n_b}
    - {click: ':'}
    - {click: '/'}
    - {click: '±'}
    - {click: '0', key_back_color: k_n_b, hilited_key_back_color: h_k_n_b}
    - {click: '.'}
    - {click: ','}
    - {click: '='}
    - {click: Keyboard_default, long_click: Keyboard_symbols}
    - {click: space}
    - {click: BackSpace}
    - {click: Return}
#步骤二，在配色方案中定义k_n_b和h_k_n_b的颜色
"preset_color_schemes/xxx": #配色方案，直接利用了上一节示例做的配色
  name: xxx极简
  back_color: 0xEEF1E7
  text_color: 0x000000
  key_back_color: 0x60DEEDB1
  hilited_candidate_back_color: 0x80D4ED89
  keyboard_back_color: xxx.jpg
  k_n_b: 0x80D4ED89 #数字键背景色
  h_k_n_b: 0x60DEEDB1 #高亮数字键背景色
```

好了，看看效果：

+	1	2	3	#
-	4	5	6	%
*	7	8	9	:
/	±	0	.	,
=	返回	自然码++	退格	Enter

※自定义的分组标签可以在当前主题的所有配色方案中使用。只需要在相应的配色方案中给 `k_n_b` 和 `h_k_n_b` 设定颜色值即可。这样每个配色方案中的数字键都可以独立设置颜色，自由度更高。（若某个配色方案中的 `k_n_b` 省略不写，则数字键的背景色会默认使用普通按键背景色，不用担心会出错。）

分组标签的使用还可以更灵活。

例4：分组标签与 `fallback_colors` 配合

细心的你可能会发现，例3中的数字键颜色，只是把普通按键的背景与高亮背景翻转过来而已。像这样很有规律的对应关系，使用 `fallback_colors` 会更简单。

#将这两句添加到例3的补丁中（在 `fallback_colors` 中建立自定义分组与普通按键的对应关系）

```
"fallback_colors/k_n_b": hilited_key_back_color #使数字键的背景色=普通按键的高亮色
"fallback_colors/h_k_n_b": key_back_color      #使数字键的高亮色=普通按键的背景色
```

#再删掉配色方案中的这两句（配色方案中的 `k_n_b` 和 `h_k_n_b` 可有可无，若存在则优先使用）

```
k_n_b: 0x80D4ED89 #数字键背景色
h_k_n_b: 0x60DEEDB1 #高亮数字键背景色
```

这样就进一步简化了配色方案。

咦，怎么感觉绕来绕去的，为什么不直接在键盘上写 `{click: '1', key_back_color: hilited_key_back_color, hilited_key_back_color: key_back_color}` 呢？其实这样写也可以，例2就是这样做的。

例4是综合了例2、例3的优点。在 `fallback_colors` 中建立对应关系，可以简化配色方案，另外还给配色方案单独控制数字键颜色预留了一个通道，可以实现更多的可能。

（若您的主题中只有一个配色方案，那基本上不需要这些复杂的方法）

四、 `android_keys`

这一部分列出了所有已知的按键以及各种可用的条件、功能。目前的主要作用是供我们查阅，在后面定义 `preset_keys` 和 `preset_keyboards` 时会用到这些。

- `name`：罗列了已知按键的名称，每个名称都代表一种按键功能。
- `when`：按键功能的各种触发条件
 - `ascii`：西文标签（处于英文状态时）
 - `paging`：翻页标签（翻页时）
 - `has_menu`：菜单标签（出现候选项时——非空码时）
 - `composing`：输入状态标签（处于输入过程中）
 - `# always`：始终
 - `# hover`：滑过
 - `click`：单击
 - `long_click`：长按
 - `combo`：并击
 - `# double_click`：双击
 - `swipe_left`：左滑
 - `swipe_right`：右滑
 - `swipe_up`：上滑

- `swipe_down` : 下滑
※标注#的暂未实现。
- `property` : 各种属性
 - `width` : 宽度
 - `height` : 高度
 - `# gap` : 间隔
 - `preview` : 按键气泡提示
 - `hint` : 按键助记 (用于显示双拼的韵母等)
 - `label` : 按键标签
 - `text` : 组合键 (用于输出各种组合键)
 - `states` : 状态标签
 - `repeatable` : 长按重复
 - `functional` : 功能键
 - `shift_lock` : Shift键锁定方式 (`click` : 单击锁定, 可用于「选择」键; `long` : 长按锁定; `ascii_long` : 仅英文状态长按锁定)
- `action` : 执行的动作
 - `command` : 执行命令 (输出时间等)
 - `option` : 命令参数
 - `select` : 选择 (键盘布局)
 - `toggle` : 切换状态
 - `send` : 发送按键

※ `when name` 的用法可参考[表1 按键功能组合示例](#)。

※其它参数的用法可参考默认的 `trime.yaml` 。

`android_keys` 极少需要改动, 用户自定义的主题可直接导入默认主题的对应节点:

```
#xxx.trime.yaml
android_keys:
  __include: trime:/android_keys #导入trime.yaml中的android_keys
```

以后默认主题若变更了 `android_keys` , 自制的主题会自动跟进。这样可以节省维护主题的时间。

五、 `preset_keys`

按键预定义。在这里对功能键进行添加、删除、重定义等操作。

※默认 `trime.yaml` 的 `preset_keys` 预置了很多功能键 (比如语音输入、撤销&重做、切换键盘、运行程序、搜索字符串等等), 在定制键盘布局时需要到这里查阅 (回厂后的 `trime.yaml` 内有详细的注释, 这里不再赘述)。

示例: 调整按键属性

例1: 更改回车键标签

在 `trime.yaml` 中, 回车键默认是这样定义的:

```
Return: {label: Enter, send: Return}
```

展开来是这样:

```
Return:
  label: Enter
  send: Return
```

由 `label: Enter` 可知: 回车键默认显示的标签是 `Enter`。

如果要把回车键显示成 `回车`, 可以这样:

```
# trime.custom.yaml
patch:
  "preset_keys/Return":
    label: "回车"
    send: Return
```

例2：更改空格键的按键气泡提示

如果输入方案的名字很长，空格键的按键气泡也会非常长。可以通过定义空格键的 `preview` 属性来解决这个问题。

对于一般的按键：

- 如果没有设置 `preview`，那么同文会以按键的标签 `label` 来做按键气泡提示。
- 如果也没有设置 `label`，那么会以按键所执行的命令及其相关的状态来做气泡提示。（对于空格键来说，会使用输入方案的名字）。

默认的空格键是这样：

```
space: {repeatable: true, functional: false, send: space}
```

我们给它添加一个 `preview` 属性

```
# trime.custom.yaml
patch:
  "preset_keys/space":
    repeatable: true
    functional: false
    preview: " " #把空格键的气泡提示设为空格
    send: space
```

例3：一键输出「日期+时间」

以预设26键键盘为例：

```
# trime.custom.yaml
patch:
# 参考trime.yaml内置的date键，新建一个按键date_time
  "preset_keys/date_time":
    command: date
    label: time
    option: "yyyy-MM-dd HH:mm:ss" #通过`option`参数控制输出的日期和时间格式
    send: function

# 用date_time替换原预设26键键盘中的time
  "preset_keyboards/qwerty/keys/@26/long_click": date_time
```

※ 常用的时间选项：`y` 年，`M` 月，`d` 日，`h` 时（12小时制），`H` 时（24小时制），`m` 分，`s` 秒，`S` 毫秒，`E` 星期，`D` 一年中的第几天，`w` 一年中第几个星期，`a` 上午/下午，`z` 时区

例4：关闭功能键属性

在 `preset_keys` 里面定义的按键，默认会打开 `functional` 属性。这些按键在键盘上会显示出功能键特有的颜色（比如回车键和退格键）。假设要让回车键也变为普通按键的颜色，可以关闭它的 `functional` 属性（关闭后只会改变功能键的颜色，其它功能不会有变化）

```
# trime.custom.yaml
patch:
  "preset_keys/Return":
    functional: false #不使用功能键颜色
    label: Enter
    send: Return
```

例5：自定义组合键

使用 `text` 可以实现一些比较复杂的操作

比如：

```
overwrite: {text: "{Control+a}{Control+v}", label: 覆盖}
```

这个组合键把全选和粘贴合并起来了。按下它就可以用剪贴板中的内容覆盖当前文档。

`text` 的格式：`text: "{send|key}{send|key}....."`（功能键必须用大括号 `{}` 括起来，其它的文本或符号可以省略括号）

`text` 的其它用例：

- `text: "{Escape}/{fh}"`：清空前面的输入码并输入 `/fh`（配合 `symbols.yaml` 可以输入符号）
- `text: "「 {Left}{Keyboard_default}"`：输出成对符号「」并把光标移到符号中间再返回主键盘
- `text: "{Control+Left}"`：逐词移动。（单个组合键也可以直接用 `send: Control+Left`，`text` 可以看作是组合键的组合）
- ...

可以自由发挥想象力，看看你造出来的组合键同文能不能支持。

六、preset_keyboards

预置的键盘布局。

键盘布局

一个主题里可以有多个键盘布局。

`default` 键盘布局ID，不可重复

- `name`：布局名称
- `author`：作者信息
- `ascii_mode`：键盘的默认状态（`0`：中文；`1`：英文）
- `ascii_keyboard`：非标准键盘（比如注音、仓颉、双键等），在切换到英文模式时，自动跳转到这里设定的英文键盘（试验功能，有待完善）
- `label_transform`：中文模式下按键字母标签自动大写（`uppercase`：自动大写，仅对单个字母生效，长标签请直接更改`label`；`none`：无，可省略不写）
- `lock`：在不同程序中切换时锁住当前键盘，不返回默认的主键盘。用于单手键盘等。（`true`：锁住；`false`：不锁，可省略不写）
- `columns`：键盘最大列数，超过则自动换行，默认30列。
- `width`：按键默认宽度（也可以在按键里面单独定义某个按键的宽度）
- `height`：每行的高度（要想改变单独一行的高度，可以直接在那一行行首的按键里设 `height`）
- `key_hint_offset_x`：助记符号x方向偏移量（向右为正，下同）
- `key_hint_offset_y`：助记符号y方向偏移量（向下为正，下同）
- `key_symbol_offset_x`：长按符号x方向偏移量
- `key_symbol_offset_y`：长按符号y方向偏移量
- `key_text_offset_x`：按键文本x方向偏移量
- `key_text_offset_y`：按键文本y方向偏移量
- `key_press_offset_x`：按键按下时所有文本x方向偏移量
- `key_press_offset_y`：按键按下时所有文本y方向偏移量
- ※以上这几个offset也可以直接写在按键中，仅对该按键生效。
- `keys`：按键排列顺序
键盘中每对`{}`括号代表一个按键，按从左到右、从上到下的顺序排列。每行的宽度排满 100 或虽然不足 100 但无法再容纳一个按键又或者每行按键数量达到 `columns` 的设定值时，转到下一行继续排列。

布局调用

`trime.yaml` 已经内置了很多种键盘布局，一般常用的输入方案都可以自动匹配到合适的预置键盘。

※ `style/keyboards` 中的 `.default`，就是用来自动匹配键盘布局的。

自动匹配的过程：

- 如果输入方案的 `schema_id` 可以找到对应的键盘布局 ID，则直接使用这个布局
比如仓颉五代的 `schema_id` 是 `cangjie5`，在 `trime.yaml` 中刚好有 ID 为 `cangjie5` 的键盘布局，那就直接使用它。
- 如果匹配不了 ID，那根据输入方案的 `speller/alphabet` 所用的字符，匹配最合适的布局方案
比如朗月拼音的 `speller/alphabet` 是 `zyxwvutsrqponmlkjihgfedcba`，恰好使用了26个英文字母。那就自动套用 预设26键 键盘。

- 如果 ID 和 speller/alphabet 都匹配不到，就用默认的 预设26键 键盘。

如果自动匹配的布局不理想，还可以手动设置。如下面的示例。

示例：指定朗月拼音使用36键键盘布局

(36键键盘比26键的多了一排数字键，可以快速输入数字)

```
# trime.custom.yaml
patch:
  "preset_keyboards/luna_pinyin/import_preset": qwerty0  #预设36键布局的ID是qwerty0
```

如是即可。

再看看，重新部署后，补丁融入 trime.yaml 之中，就被展开成这种格式：

```
luna_pinyin:
  import_preset: qwerty0
```

可以理解成：新建了一个 ID 是 luna_pinyin 的布局，这个布局导入了 qwerty0 的全部设置。

△ 如果这里指定的键盘出错了，就会自动调用 default 键盘。

布局调整

键盘布局就像积木一样，是由各种功能组合&排列而成。

先来看看按键是怎么由一个个功能组合而成的：

表1 按键功能组合示例

No.	按键	功能
1	{click: g}	单击时输出 g ，没有其它功能。
2	{width: 5}	这是一个宽度为5的空白间隙。
3	{click: space, width: 28}	单击输出空格，按键加宽（到 28 ）。
4	{click: h, long_click: ""}	单击输出 h ，长按输出 ‘ 撇号 ’。
5	{click: e, label: '水', long_click: '+', label: '+'}	单击输出 e ，在中文状态时按键标签是 水 ，英文时标签恢复成 e ，长按输出 + 。用于仓颉键盘。
6	{click: v, long_click: '~', swipe_left: Date, swipe_right: Time}	单击输出 v ，长按输出 ~ ，左滑输出日期，右滑输出时间 。
7	{click: Shift_L, composing: "", width: 15}	平时单击切换大小写，在打字过程中变为分词键 '。按键加宽到 15 。
8	{click: '.', long_click: '>', has_menu: '次选', send_bindings: false}	单击输出 . ，长按输出 > ，打字出现候选时按键标签变为「次选」。△ send_bindings 用来控制 composing 、 has_menu 、 paging 时是否发送按键给后台（ true ：发送； false ：不发送，仅改变按键标签，按键的实际功能仍是 click ）。 send_bindings 默认为 true ，可以省略不写。
9	{click: '.', long_click: '>', has_menu: Page_Down}	平时单击输出句点 . ，长按输出 > ，打字出现候选项时，变为 向下翻页键 。
10	{click: '<', long_click: '<', paging: Page_Up}	平时输出 < ，长按输出 < ，翻页时变为 向上翻页键 。
11	{click: "ㄅ", ascii: g}	单击时输出符号“ㄅ”，在英文状态下输出 g
12	{click: "h", hint: "ang"}	单击输出 h ，在按键下方显示韵母 ang ，用于双拼等助记键盘
13	{click: "({Left}")}	单击时输出一对括号 () ，且光标自动移到括号中间。※与 preset_keys 里面的 text: "({Left})" 等效，但 text 不能直接用在键盘布局中，要改成 click 、 long_click 等
14	{click: "q", height: 60}	单击时输出 q ，当该键位于行首时，整排按键加高到60

15	<code>{click: ""}</code>	这是一个空按键，按下去不会触发任何动作。空键的其它写法： <code>click: "VoidSymbol"</code> ， <code>composing: "VoidSymbol"</code>
16	<code>{click: Return, combo: g}</code>	单击时是回车键，与其它按键并击时输出 <code>g</code> 。※ <code>combo</code> 通常用于并击方案，可以复用一些功能键（比如 <code>space</code> 、 <code>Keyboard_number</code> ），节约空间。△ 因 <code>combo</code> 与 <code>repeatable</code> 属性有冲突，所以类似退格键这样的按键必须关掉 <code>repeatable</code> 才能使用 <code>combo</code>

我们在 `android_keys` 和 `preset_keys` 中提及的触发条件和按键，都可以按这种格式(`when: name, when: name, when: name, ……`)组合起来，还可以加上一部分的 `property`（比如 `width` 和 `label`）。基本上就是这样来做组合了。

※如果指定的 `name` 在 `android_keys` 和 `preset_keys` 中都找不到，那就以文本形式直接输出（比如 `{click: 你好}`），单击该键时，就直接输出「你好」。在制作特殊符号键盘时，可能需要这种效果。

按键造好了，再按一定的顺序排列起来就成了布局。

示例：给键盘添加删词功能

对于使用 `script_translator` 的拼音类输入方案，如果在打错词后，马上按退格键删除已经上屏的错词，可以使错词不被记录到用户词典中。但是如果隔的时间太长，或者使用的是 `table_translator` 形码方案，那就没办法这样删词了。这时候就需要定制键盘来辅助我们进行删词。

以明月拼音为例，键盘示意图：



※ 由于用到了 `has_menu` 条件，键盘右上角的左右方向键和删词键只在正常打字的过程中出现。处于英文状态或不打字时仍然是数字键8、9、0。

```
# trime.custom.yaml
patch:
# 1、让明月拼音使用36键键盘布局
"preset_keyboards/luna_pinyin/import_preset": qwerty0

# 2、给36键键盘添加方向键和删词键
"preset_keyboards/qwerty0/keys/@7/has_menu": Left
"preset_keyboards/qwerty0/keys/@8/has_menu": Right
"preset_keyboards/qwerty0/keys/@9/has_menu": DeleteCandidate
```

△ 与 PC 版的 Rime 一样，只能从用户词典中删除词组，若要删除固态词典中的词组，请直接修改 `dict.yaml` 文档。

示例：新建一个副键盘

```
# trime.custom.yaml
patch:
# 1、新建一个键盘布局xkey
"preset_keyboards/xkey": #布局ID
  author: "xq" #作者
  name: "数字+操作键" #方便自己辨识的名字
  ascii_mode: 1 #默认进入英文状态
  height: 44 #每行高度
  width: 11 #按键默认宽度（取所有按键中用得较多的宽度值，接下来就可以少写一些width了）
  keys: #按键排列
    # 第一行
    - {click: "#", long_click: cut}
    - {click: "%", long_click: copy}
    - {click: "@", long_click: paste}
    - {click: "+", width: 13}
    - {click: 7, width: 18}
    - {click: 8, width: 18}
    - {click: 9, width: 18}
    # 第二行
    - {click: Page_Up}
    - {click: Up}
    - {click: Page_Down}
    - {click: "-", width: 13}
    - {click: 4, width: 18}
    - {click: 5, width: 18}
    - {click: 6, width: 18}
    # 第三行
    - {click: Left}
    - {click: Tab}
    - {click: Right}
    - {click: "x", width: 13}
    - {click: 1, width: 18}
    - {click: 2, width: 18}
    - {click: 3, width: 18}
    # 第四行
    - {click: Home}
    - {click: Down}
    - {click: End}
    - {click: "÷", width: 13}
    - {click: ",", width: 18}
    - {click: 0, width: 18}
    - {click: BackSpace, width: 18}
    # 第五行
    - {click: Keyboard_default, long_click: Menu, width: 18}
    - {click: "±", width: 14.9} #如果上下的按键没有严格对齐，可以微调宽度
    - {click: "=", width: 13}
    - {click: space, width: 18}
    - {click: ".", width: 18}
    - {click: Return, width: 18}

# 2、要在主键盘中调用它，就得新建一个功能键，来开启这个ID为xkey的键盘
"preset_keys/Keyboard_xkey": #按键ID
  label: 123 #按键标签
  send: Eisu_toggle #执行切换键盘命令
  select: xkey #选择xkey键盘

# 3、把这个切换键盘的功能键放到主键盘中（以预设26键键盘为例）
"preset_keyboards/qwerty/keys/@31/long_click": Keyboard_xkey #通过长按符号键来开启这个键盘

# 4、最后在style/keyboards中声明一下我们要用到xkey这个键盘
"style/keyboards":
  - .default
  - default
  - number
  - symbols
  - xkey #style/keyboards不能只写xkey一个，其它用到的键盘要照原样抄过来，不然会出错
```

副键盘就这样做出来了。效果图：



△ 如果是新建主键盘，则可以省略步骤2、3、4，因为同文可以根据键盘ID自动调用键盘。

常见问题：修改不生效？

比较常见的原因是配置文件出现语法错误了，需要检查空格缩进、引号配对、是否错用tab等等。哪怕是一个空格出错了，也有可能使整个配置文件失效，初学者需要格外小心。

仅供参考，欢迎指正。

附录： schema.yaml 中的 trime

还可以针对不同输入方案设置主题参数。

1、 style

在 schema.yaml 里面设置的 style 参数会优先生效。

示例：为英文输入方案设置专用的英文字体

```
#easy_en.schema.yaml

style:
  candidate_font: gunplay.ttf #相应的字体文件需放在rime/fonts
  comment_font: gunplay.ttf
  #也可以直接设置latin_font，但latin_font暂时对悬浮窗不起作用
```

当切换到easy_en方案时，候选栏的字体自动变为gunplay。（在使用其它输入方案时，仍然使用 trime.yaml 里面设定的字体）

2、 switches

在schema里面可以给这些开关设置初始状态：

- `_hide_candidate` 隐藏候选栏
- `_hide_comment` 隐藏候选项注释
- `_hide_key_hint` 隐藏按键助记符号

※ 这几个开关分别对应 preset_keys 里的功能键：`Candidate_switch`、`Comment_switch`、`Hint_switch`。

示例：盲打方案自动关闭候选栏

```
#xxx.schema.yaml

switches:
  - name: _hide_candidate
    reset: 1 #默认开启这个开关，隐藏候选栏
    # 由于没有设置states，这个开关是隐藏的。只在切换到该方案时将_hide_candidate设为1。
```

当切换到这个方案时，候选栏自动关闭。

※在需要时，也可以通过快捷键或键盘按键随时打开候选栏。

也可以在状态栏上添加按键：

示例：在状态栏放置按键（试验功能）

option 格式：

- `_key_ + 按键`（这里的按键可以是功能键、符号、字母、任意文本）
- `_keyboard_ + 键盘布局ID`（一些比较特殊的id：`.default` 返回当前主键盘，`.next` 下一个键盘，`.last` 在最近使用的两个键盘间轮换，`.last_lock` 返回上一个标记为 lock 的键盘）

```
#xxx.schema.yaml

switches:
  # 这个按键用来收起软键盘
  - options: [ _key_Hide ]
    states: [ "▼" ]
  #states是显示在状态栏的图标，options是实际执行的功能

  # 这个按键用来调出输入法切换菜单
  - options: [ _key_IME_switch ]
    states: [ "□" ]
  # 更多功能键请查阅trime.yaml/preset_keys

  # 这是一个空按键，按下去不会执行任何动作
```

```
- options: [ _key_VoidSymbol ]
  states: [ " " ] #空键的宽度由这里的空格决定

# 这个按键用来输出句号
- options: [ _key_period ] #period是句号的英文名，也可以直接写成[ _key_. ]
  states: [ "." ]

# 这个按键用来输出固定的短语
- options: [ _key_吃饭了吗? ]
  states: [ "☐" ]

# 这个按键用来切换键盘布局（在主键盘、数字键盘、英文键盘之间轮换）
- options: [ _keyboard_default, _keyboard_number, _keyboard_letter ]
  states: [ 默认, 123, abc ]
#多个键盘轮换的功能还有一些问题，有待完善

# 也可以只切换到一个特定的键盘
- options: [ _keyboard_number ]
  states: [ 123 ]

#...
```

trimer小知识(1)---Yaml文件开头注释是什么意思？

编辑小狼毫的方案及配置项文件时，通常会看到这样文件开头：

```
# wubi86_double_key.yaml
# vim: set sw=2 sts=2 et:
# encoding: utf-8

schema:
  schema_id: wubi86_double_key
  name: "五笔86双键版"
....
```

那么开头的注释有什么含义呢？

第一行详解

第一行比较简单，一般要放文件名，或者其他注释，不要放正式内容。为了防止加了Bom的utf-8文件无法解析。具体原因见, [Rime 輸入方案設計書](#)

鑑於一些文本編輯器會為 UTF-8 編碼的文件添加 BOM 標記，為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

因为默认的文件存储格式是utf-8,而utf-8又分带bom和不带bom两种格式。带bom会在开头添加几个字节，方便程序判断一个文本是否为utf-8编码。

如果你是程序员，要编写程序读写utf-8,还可以看看这里的bom详解：[UTF8最好不要带BOM，附许多经典评论](#)

第二行详解

正式开始最重要的一行，就是第二行

```
# vim: set sw=2 sts=2 et:
```

这是什么意思呢？

这一行有个学名叫modeline，是vim专用的。用vim打开这个文件时，会自动运行该命令，设置好阅读和编辑该文件的一些参数

[StackOverflow上关于modeline的解释](#)

命令的具体含义，参看下面的解答

```
"vim中每个命令都是简写和全拼两种模式，后面列出命令的全拼，大家就知道什么意思了
set sw=2 "sw即shiftwidth,设置自动缩进 2 个空格
set sts=2 "即设置 softtabstop 为 2. 输入 tab 后就跳了 2 格
set et "设置expandtab,即将tab扩展为空格,如果要取消这个选项，为 :set noet
" vim的开头命令，都是在前面加no表取消
```

命令之间是通过空格或者":"分隔的，最后那个":"起分隔作用，表示设置结束

所以，总结一下就是，编辑yaml文件的具体环境为：

- 自动缩进为2
- tab键缩进相当于2个空格
- 将tab键自动扩展为空格

当然，也可以把上面的命令写在_vimrc中，作为全局设置。这样，打开编辑其他的，没有带modeline的文件时，也可以使用统一的设置。

更多的vim设置

vim中还有一些其他的缩进相关的设置也列在这里

```
set tabstop=4 "实际的 tab 即为 4 个空格, 而不是缺省的 8 个

# 设置自动的缩进风格
set ai "设置自动缩进
set cindent "设置使用 C/C++ 语言的自动缩进方式
```


trimer小知识(1)---Yaml文件开头注释是什么意思？

关于Vim的tabstop,softtabstop的区分，以及与shiftwidth，expandtab组合使用的具体含义。参见下面的帖子

[vim中tabstop、shiftwidth、softtabstop以及expandtab的关系](#)

从上面，我们可以看出列表和映射的两种表示方法：

列表的单行与展开模式：

```
# 单行模式:
name: [value1, value2]
# 展开模式:
name:
- value1
- value2
```

映射的单行与展开模式：

```
# 单行模式:
name: {p1: n1, p2: n2}
# 展开模式:
name:
  p1: n1
  p2: n2
```

数据引用方式

打patch时，需要先取得准备修改的属性，然后才能对其进行修改。

参见晓群老师，给我的示例，

```
"preset_keyboards/qwerty/keys/@31": {label: "英", click: Keyboard_default}
```

总结如下：

列表数据的引用：

通过@下标，引用。

映射数据的引用：

"/"一层一层的引用

关于@的更多语法，见 [定制指南](#)

```
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（不建議在補訂中使用）
  "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（不建議在補訂中使用）
  "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（不建議在補訂中使用）
```

打补丁注意事项

打补丁，相当于对某个数据域重新进行赋值。该数据域如果是一个映射，那么它原始值中未被赋值的属性就会被删除。所以，要精确引用到需要进行修改的数据域很重要。如果引用的数据范围过大，而又没有给其中的所有属性赋值，部署时，程序由于读不到某些关键的属性，就会崩哦。

比如，现在需要定制输入方案，切换到英文模式时，调用标准的qwerty键盘。但是需要对qwerty键盘打补丁，对其中的某些项作修改。

```
# 对键高度，和某个按键的事件，以及中/英模式，做补丁修改

# 方法1，OK，可以正常工作，精确指定了修改的属性
"preset_keyboards/qwerty/height": 60
"preset_keyboards/qwerty/keys/@31": {label: "英", click: Keyboard_default}
"preset_keyboards/qwerty/ascii_mode": 1

# 方法2，用全覆盖的方式定制
```

```
"preset_keyboards/qwerty": #注意这里，相当于对整个qwerty重新赋值
  height: 60 #每行的高度
  keys: #这里必须重新把所有要使用的按键都声明一遍
    - {click: space}
  ascii_mode: 1
```

#注：用这种方式定制时，相当于对整个qwerty赋值，不能只赋值其中的一部分，而是要全部赋值。否则，未出现的属性值，会被删除

其他

另外，网上还有数据引用和合并的介绍，但是我还没有验证，在我们程序中是否可行。

可以使用&符号定义一个引用标签，使用符号*引用这个标签的数据，使用符号<<进行hash值合并操作，例如：

```
# sequencer protocols for Laser eye surgery
---
- step: &id001 # defines anchor label &id001
  instrument: Lasik 2000
  pulseEnergy: 5.4
  pulseDuration: 12
  repetition: 1000
  spotSize: 1mm
- step:
  <<: *id001 # merges key:value pairs defined in step1 anchor
  spotSize: 2mm # overrides "spotSize" key's value
- step:
  <<: *id001 # merges key:value pairs defined in step1 anchor
  pulseEnergy: 500.0 # overrides key
  alert: > # adds additional key
    warn patient of
    audible pop
```

1. &id001定义了一个id001的引用标签（引用文档中第一个step元素的所有属性）；
2. 第二个step元素引用id001后，重写spotSize属性；
3. 第三个step元素引用id001后，重写pulseEnergy属性，并添加alert属性

五笔双键配置案例详解(一)-准备篇

五笔双键配置案例详解(一) 准备篇

这是我认识同文和小狼毫以来，配置的第一个输入方案。感谢全程有@xiaoqun2016和@osfans老大的细致而耐心的指导。记下整个过程，方便后来者学习。通过本案例，你可以学到：

- 小狼毫的一般配置和调试流程
- 小狼毫模糊音的使用
- 同文键盘的配置和对已有键盘布局的修改
- 键盘的切换

一般的调试流程

"工欲善其事，必先利其器"。搭建好下开发环境，使用良好的配置工具，做事可以事半功倍。

文本编辑器

我使用的是gVim，在Windows下。Vim编辑yaml文件有很多好处：

- Vim自带了yaml文件的语法高亮，方便查看
- 小狼毫自带的yaml文件开头都配有vim的modeline行，会自动设置好缩进等相关选项，方便编辑
- Vim本身也很强大，使用范围广，教程多，而且开源免费

调试流程

1. 将你要反复修改或者反复打patch修改的yaml文件复制一份，放到你的工作目录(这样防止修改错了需要回厂)
2. 在工作目录创建XX.custom.yaml文件(如果已有就不要创建了) -- 建议不要在原文件上修改,而是通过打patch方法修改
3. 有两个批处理脚本如下,你可以直接复制过去使用

```
:: rime.bat
copy wubi86_double_key.schema.yaml %appdata%\rime
copy wubi86.dict.yaml %appdata%\rime

start WeaselDeployer.exe /deploy ::重要的是这句，需要将小狼毫的安装目录添加到path环境变量中

rem 或者写成这样
rem "C:\Program Files\Rime\weasel-0.9.14\WeaselDeployer.exe" /deploy
pause
```

如果是在手机上调试同文输入法，则使用下面的：

```
:: trime.bat
adb push trime.yaml /sdcard/rime
adb push trime.custom.yaml /sdcard/rime
adb push wubi86_double_key.schema.yaml /sdcard/rime
adb shell am broadcast -a com.osfans.trime.deploy
pause
```

这样做的好处，就是每次修改，都相当于把系统文件恢复到你打patch之前的状态，你打patch造成的错误不会累积。

定制输入方案

我要实现的键盘效果如下所示：



实现效果类似九宫格，但是因为一个键上只有两个键同时又保留了电脑上的键盘布局。所以，保留了五笔的输入模式。

大致的步骤是（这也是本系列文章的标题）：

1. 先复制wubi86的输入法方案,实现自己的方案
2. 添加模糊音，实现双键方案
3. 对双键方案进行调整完善（码表共享和词频调整）
4. 在手机上定义双键键盘，实现中文输入
5. 定义键盘切换，实现英文输入
6. 对输入方案优化（符号选2-3候选字等等）

1 – 3都是在电脑的小狼毫进行调试，4 – 6则是在手机的同文输入法上进行调试。

五笔双键配置案例详解(三)-用模糊音实现双键转换

前面的热身已经结束，下面开始真正的定制了。

这部分的定制还是在PC的小狼毫上进行，主要修改上一节中复制得来的，wubi86_double_key.schema.yaml文件。

具体的配置

将wubi86_double_key.schema.yaml文件中的speller节点修改如下：

```
speller:
  delimiter: " ;'"
  max_code_length: 4
  algebra:
    - derive/w/q/
    - derive/r/e/
    - derive/y/t/
    - derive/i/u/
    - derive/p/o/

    - derive/s/a/
    - derive/f/d/
    - derive/h/g/
    - derive/k/j/

    - derive/c/x/
    - derive/b/v/
    - derive/m/n/
```

上面的代码就是用来改造五笔双键，也是双键输入法定制的核心。

原理说明

这里主要用到的技术就是模糊音。 注意上面的derive你可以理解为替换的意思，比如 - derive/w/q 就是将码表编码中的所有w都替换为q。

从我们要定制输入法图片上，我们可以看出，五笔双键的键盘每个按键上有两个键。 但是，目前小狼毫的libime并不支持，输入时对按键进行模糊化，即输入一个按键可能有两种情况（也正是这个原因，同文输入法目前不能支持九宫格，期待@佛振老大等人可以快点更新呀）。

我们不能对输入编码进行模糊化，那就只能对码表进行模糊化。

想象一下，我们现在只能输入一半的按键，比QW键其实是Q键，只是外观写的是"Q/W"。这样用户就只能输入Q，而不可能输入W了。 如果用一个简单的直接的办法实现，能将码表中所有的字都打出来，那就是事先将码表中的所有的W都替换成Q。将R替换成E等等。这样就可以实现模糊化了。 因为，输入Q时候，原来为编码为W的字(编码已经被替换成Q)也会出来。 这样要修改码表，虽然比较复杂，也不推荐，但是可以帮助我们理解一下概念。

拼写运算是rime对输入法编码进行定制的最强大的武器。 而正则表达式又是拼写运算的核心知识点。 关于这两者，大家有兴趣可以查看设计书中具体链接。

使用和存在的问题

现在再运行脚本，部署一下。看看会发生什么？

Ctrl+~,切换到"五笔双键"。按一下A,"工"和"要"都出来了，即A和S上的简码都出来了。

呵呵，是不是实现了？ 别着急。目前还是很多问题的：

1. 现在切换到wubi86，试着输入几个字，你会发现，wubi86也被模糊化了。
2. 注意词频是有问题的，你试着输入lqtt. 显示 "输入"这个单词，居然在"加尔各答"后面？

以下都是@xiaoqun2016 老师给出的回答.具体见 <https://github.com/osfans/trime/issues/61>

共享码表的问题

在translator结点下，添加如下定义 prism: wubi86_double_key 替换prism这个属性，表示生成的码表棱镜文件。即最终会生成一个名为wubi_double_key.prism.bin的二进制文件，用来存储我们加了模糊音替换后的wubi86的码表。 如果不加这个选项，就会与原来的wubi86.schema共用一个码表，这样就会改变原来的输入方案。即使用标准的wubi86也会有模糊音的现象。

模糊化后的词频问题

这里主要涉及码表翻译器(table_translator)和音节翻译器(script_translator) table_translator对拼写运算支持得不好,经典模糊音转换来的字词的频率都默认为0,所以无法按正常的词频显示。script_translator不存在这个问题,但是可能会有别的问题。这个我也在理解中

修改后的源码

以下为我现在使用的 wubi86_double_key.schema.yaml 的源码

```
# Rime schema settings
# vim: set sw=2 sts=2 et:
# encoding: utf-8

schema:
  schema_id: wubi86_double_key
  name: "五笔86双键版"
  version: "0.12"
  author:
    - 發明人 王永民先生
  description: |
    五筆字型86版
    碼表源自 ibus-table
    請安裝【袖珍簡化字拼音】以啓用 z 鍵拼音反查

switches:
  - name: ascii_mode
    reset: 0
    states: [ 中文, 西文 ]
  - name: full_shape
    states: [ 半角, 全角 ]
  - name: extended_charset
    states: [ 通用, 增廣 ]

engine:
  processors:
    - ascii_composer
    - recognizer
    - key_binder
    - speller
    - punctuator
    - selector
    - navigator
    - express_editor
  segmentors:
    - ascii_segmentor
    - matcher
    - abc_segmentor
    - punct_segmentor
    - fallback_segmentor
  translators:
    - punct_translator
    - reverse_lookup_translator
#   - table_translator
    - script_translator

speller:
  delimiter: " ;'"
  max_code_length: 4
  algebra:
    - derive/w/q/
    - derive/r/e/
    - derive/y/t/
    - derive/i/u/
    - derive/p/o/

    - derive/s/a/
    - derive/f/d/
    - derive/h/g/
    - derive/k/j/
```


- derive/c/x/
- derive/b/v/
- derive/m/n/

translator:

dictionary: wubi86_double_key
enable_charset_filter: true
enable_encoder: true

reverse_lookup:

dictionary: pinyin_simp
prefix: "z"
tips: (拼音)
preedit_format:
- xform/([nljqxy])v/\$1ü/
- xform/([nl])ue/\$1üe/
- xform/([jqxy])v/\$1u/

punctuator:

import_preset: default

key_binder:

import_preset: default

recognizer:

import_preset: default
patterns:
reverse_lookup: "^z[a-z]*\$"

五笔双键配置案例详解(二)-添加一个输入方案

五笔双键配置案例详解(二) 添加一个输入方案

把大象装到冰箱里分几步？

答：三步：

1. 把冰箱门打开
2. 把大象推进去
3. 把冰箱门关上

添加一个输入方案分几步？

答：也是三步：

1. 得到一份方案 (编写或者复制)
2. 为输入方案定义schema_id和name
3. 将输入方案schema_id添加到default.yaml的shcema_list中

OK,现在按Ctrl+~就可以切换了。

得到输入方案

由于五笔双键是在wubi86的基础上的，所以，你必须有原始的wubi86.schema.yaml和wubi86.dict.yaml这两个文件。wubi86.schema.yaml存放的是wubi的方案的一般处理流程，我们有了这个模板，就可以方便地进行修改。wubi86.dict.yaml这个文件存放的是wubi86的码表，即所有的汉字、词组和对应的编码。我们会在这个码表的基础上进行模糊音处理，从而生成一个新的，适合双键的码表。

闲话少说，总之第一步就是原来的wubi86.schema.yaml复制一份，重命名为wubi86_double_key.schema.yaml

生成新的输入方案

第二步，就是在前一步基础上，将wubi86_double_key.schema.yaml文件中下面两项给替换掉：

```
schema_id: wubi86_double_key  #id是给系统看的，系统中引用都是通过它
name: "五笔86双键版"         #name是给人看的，切换输入方案的时候，这个名字就会出现
```

说明：替换schema_id，是因为每个输入方案必须有一个独立的标识（后面会用到）。替换了name，是给人看的。

添加方案到系统列表

经过上面的两步，我们其实就已经有了一个名字独一无二的输入方案。虽然输入主案的内容与wubi86一模一样，但是至少我们先配置一下，然后就可以在系统方案中看到它。在default.custom.yaml文件添加我们的方案id。

```
patch:
  schema_list:
    - schema: wubi86
    - schema: wubi86_double_key  #我们的方案ID
```

重新部署

现在切换到小狼毫，然后按下小狼毫的热键，Ctrl+` 或者F4试试 现在是不是就可以看到我们的输入方案的名称了？切换过去试一下，是不是跟系统自带的五笔一模一样？

提高篇

如果你有兴趣深究，还可以切换到%AppData%/rime路径下，看看重新部署后，到底生成了哪些文件。其实生成的文件中以wubi86开头的文件，只有三个：

- wubi86.prism.bin #输入法的棱镜文件
- wubi86.reverse.bin #输入法反查的文件
- wubi86.table.bin #输入法的码表文件

由于我的码表文件共用的是wubi86的文件，所以我们跟wubi86其实是共用的一套输入法棱镜文件。看wubi86_double_key.yaml中的内容，就知道了。

```
translator:  
  dictionary: wubi86  #注意这里,其实引用的是wubi86.dict.yaml码表文件  
  enable_charset_filter: true  
  enable_encoder: true
```

小结

现在，你可以回顾一下前面学到的知识。

1. 每个输入方案，有两个名字: schema_id和name。一个是系统使用，一个是显示给人看。
2. 定义了输入方案和schema_id后，只要将shcema_id添加到default.custom.yaml文件的schema_list中。就可以切换到该输入方案了。

五笔双键配置案例详解(四)-实现手机上的双键键盘

现在我们已经有了一个可以实现双键的输入方案。虽然是在电脑上测试的，但是在手机上也是可以使用的。我们还缺少一个对应的手机键盘。下面的工作需要手机上布署测试了。

具体的配置

按惯例，还是先贴上相关代码，再慢慢分析。我们通过对trime.yaml打patch来实现

```
#trime.custom.yaml
patch:
  #1、新建一個按鍵佈局wubi86_double_key
  "preset_keyboards/wubi86_double_key": #佈局ID
    author: "bobolqiqi" #作者
    name: "五笔双键" #方便自己辨識的名字
    ascii_mode: 0 #默認進入中文狀態
    height: 60 #每行高度
    width: 20 #按鍵默認寬度（取所有按鍵中用得較多的寬度值，接下來就可以少寫一些width了）
    keys: #按鍵排列
      #第一行
      - {label: "Q W", click: q, long_click: 1}
      - {label: "E R", click: e, long_click: 2}
      - {label: "T Y", click: t, long_click: 3}
      - {label: "U I", click: u, long_click: 4}
      - {label: "O P", click: o, long_click: 5}

      #第二行
      - {label: "A S", click: a, long_click: 6}
      - {label: "D F", click: d, long_click: 7}
      - {label: "G H", click: g, long_click: 8}
      - {label: "J K", click: j, long_click: 9}
      - {label: "L", click: l, long_click: 0}

      #第三行
      - {label: "词", click: Shift_L, width: 10}
      - {label: "Z", click: z, long_click: "@", width: 10}
      - {label: "X C", click: x, long_click: "!"}
      - {label: "V B", click: v, long_click: "?"}
      - {label: "N M", click: n, long_click: "."}
      - {click: BackSpace}

      #第四行
      - {click: Keyboard_symbols, width: 15}
      - {click: Keyboard_qwerty, label: "全", long_click: Menu, width: 10}
      - {click: ",", width: 10}
      - {label: " ", click: space, long_click: VOICE_ASSIST, width: 30}
      - {click: "。", width: 10}
      - {label: "123", click: Keyboard_number, width: 10}
      - {label: "确定", click: Return, width: 15}

"style/keyboards":
  - .default
  - default
  - number
  - symbols
  - qwerty

"preset_keyboards/qwerty/ascii_mode": 1

"style/candidate_view_height": 36
"style/round_corner": 0.0
"style/label_text_size": 18
"style/comment_height": 14
"preset_keys/Keyboard_defaultw": #返回中文键盘
  label: 英
  select: .default
  send: Eisu_toggle
"preset_keys/space":
  functional: false
```

```
label: " "  
repeatable: false #关掉空格键重复  
send: space
```

使用说明

1. 将上面的源码存储为 trime.custom.yaml
2. 将trime.custom.yaml以及wubi86_double_key.schema.yaml都拷贝到手机的/sdcard/rime目录下
3. 到同文的主设置界面，点击重新部署

注：也可以使用准备篇中介绍的批处理脚本，记得回去复习一下哦。

现在，在选单中切换到"五笔双键"输入方案，就可以使用它了。

原理说明

主要使用到的技术如下：

- 定义一个键盘
- 声明和切换键盘
- 通过打patch修改键盘布局的属性

建议阅读@xiaoqun2016老师和@osfans老大 写的 [trime.yaml详解](#)

