

1 简介

切换到 Windows 之后，用着顺手的俄语键盘就没了。Fcitx5 自带的键盘里也没有键位完全一样的，所以我就只好手搓了。

1.1 Fcitx5 的架构

我们现在要编写的是 fcitx5 的插件。安装它的开发包之后，所有头文件都在 /usr/include/Fcitx5/* 下了，而链接的话，直接

```
find_package(Fcitx5Core REQUIRED)
target_link_libraries(rutrslit PRIVATE Fcitx5::Core)
```

这样即可。API 也好说，我们只需要提供一个能够构造输入法引擎的东西就可以了。

```
class RusTranslitEngineFactory : public fcitx::AddonFactory {
public:
    fcitx::AddonInstance *create(fcitx::AddonManager *manager) override {
        return new RusTranslitEngine(manager->instance());
    }
};
```

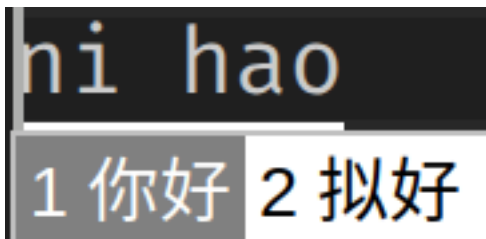
```
FCITX_ADDON_FACTORY(RusTranslitEngineFactory);
```

这个引擎会监听 fcitx5 提供给我们的事件，例如输入法激活、按键按下之类的。我们一会再来看如何将 ASCII 按键转化为西里尔字母，现在要解决的是拿到西里尔字母之后，如何让它通知应用程序。

Fcitx5 会为每个输入框都分配一个 InputContext (IC)，管理输入法在当前框内的所有状态。例如对于拼音输入法，在一个输入框里打字 sh，在另一个输入框里打字 i，并不会导致第二个输入框合成为 shi，这就是 IC 的作用。我们的编码既然并非前缀码，输入法引擎自然需要管理解码状态，而既然是状态，这个就也位于 IC 中。

在按下按键后，fcitx5 会将它转译为框架内部的 KeyEvent，并发送给 IC。接着，IC 所记录的 Instance 可能会处理/拦截这个按键，例如 ctrl + space 切换输入法。不需要拦截的按键会发送给输入法引擎，它可以决定如何处理。而它不处理的部分将直接传递给客户端。

听起来很简单，但 fcitx5 有两种不同的 UI：输入法框架自己绘制的、候选框内部的字，也就是图中的下半部分；以及在客户端（文本编辑器）里未完成的、带有下划线的字，也就是上半部分。客户端不见得都支持这个。InputContext::updatePreedit 可以将它发给客户端，而 updateUserInterface 则是修改 fcitx5 自身的 UI。



1.2 码表

我们采用如下的西里尔字母对应：

拉丁	西里尔	拉丁	西里尔	拉丁	西里尔	拉丁	西里尔	拉丁	西里尔	拉丁	西里尔
a	а	b	б	v	в	g	г	d	д	e	е
z	з	i	и	j	й	k	к	l	л	m	м
n	н	o	о	p	п	r	р	s	с	t	т
u	у	f	ф	h	х	c	ц	y	ы	w	э

'	Ъ	"	Ъ	zh	ж	ch	ч	sh	ш	sc	щ
yu	ю	ya	я	yo	ё						

这明显不是前缀码，例如 s 既可以独立成为 c，也可以和 h 一起组合为 m。那么，我们的输入法引擎自然需要将可能延长为其他码的字符留在 preedit buffer 里。如果下一个字符说明这并非前缀，我们直接将缓存的部分贪婪匹配后输出。

在按下 esc 键或者切换输入法的时候，我们就不管前不前缀的事了。总之先把已经有的东西全部输出再说，不过机制依然是贪婪匹配。这都说好。

比较麻烦的地方是到底哪些东西是 ASCII，哪些是 UTF-8。在 C++ 里全都是 std::string，两眼一抹黑，完全分不清。所以，我的解决方案是直接封装一层，分离 ascii_string 和 utf8_string，对于后者不提供 size() 和 operator[]，以免不小心从中间切开。这意味着我们的码表的拉丁字母部分必须是 ASCII（废话）。

Вот и все.