1. IME 输入

Bevy 支持 IME（输入法编辑器），这是人们在使用中文等语言进行文本输入时所使用的方法，也是非键盘文本输入方法（如手写识别）工作的方式。然而，这需要你进行一些特殊处理。

如果你希望所有国际用户能够像在其操作系统上的其他 GUI 应用程序中那样，以他们的语言输入文本，你应该支持 IME。如果你希望为残障用户或喜欢使用手写识别等替代文本输入方法的用户提供良好的可访问性，你也应该支持 IME。这应该是在支持通过键盘输入文本的基础上进行的，因为大多数用户将通过键盘输入文本。

2. IME 的工作原理

IME 通过使用一个特殊的“缓冲区”来工作，该缓冲区显示当前正在进行的文本建议，并允许用户预览和撰写文本的下一部分，然后再确认它。文本建议由操作系统提供，但你的应用程序需要为用户显示它们。

例如，假设你的 UI 中有一个文本输入框。你显示用户已经输入的文本，光标位于末尾。

如果启用了 IME，你将收到 Ime::Preedit 事件，用于“待定”文本。你应该在文本输入框中显示该“未确认”的文本，但使用不同的格式使其在视觉上有所区别。

当用户确认他们想要的输入时，你将收到一个 Ime::Commit 事件，其中包含最终文本。然后，你应该丢弃任何先前的“未确认”文本，并将新文本附加到实际的文本输入字符串中。

3. 如何在你的 Bevy 应用程序中支持 IME

首先，你需要在应用程序期望文本输入时通知操作系统。你也不希望 IME 在游戏过程中意外激活。

每当你希望用户输入文本时，你需要在窗口上启用“IME 模式”。完成后，禁用它。

如果用户没有使用 IME，启用“IME 模式”时不会发生任何事情。你仍然会像往常一样收到键盘事件，并可以通过这种方式接受文本输入。

如果用户有 IME，你将收到一个 Ime::Enabled 事件。

然后，你可以处理 Ime::Preedit 事件以获取待定/未确认的文本，并处理 Ime::Commit 事件以获取最终/确认的文本。

```
fn ime_input(mut input_text: Query<&mut Text,
With<InputText>>, mut ime_reader: EventReader<Ime>) {
    let Ok(mut text) = input_text.get_single_mut() else {
        return;
    };
    for ime in ime_reader.read() {
        match ime {
            Ime::Preedit {
                window: _,
                value,
                cursor: _,
            } => {
                text.sections[1].value = value.into();
            }
            Ime::Commit { window: _, value } => {
                text.sections[0].value.push_str(value);
                text.sections[1].value.clear();
            }
            _ => (),
        }
    }
}
```

在实际应用程序中，你将希望在屏幕上显示“预编辑”文本，并使用不同的格式显示光标。在“提交”时，你可以将提供的文本附加到通常接受文本输入的实际字符串中。