Windows 应用无障碍开发指南

发起单位：腾讯搜狗输入法、阳光读屏

参与单位：争渡读屏、NVDA 中文网、上海有人公益基金会

1 总则

近年来，随着计算机在视障群体中的不断普及，更多工作、学习、生活类的需求从线下转移到线上，但因其软件从业者信息无障碍意识较弱，相关参考文档未成体系，与此同时，产品经理对视障群体的需求不了解、不知道，开发的软件普遍存在信息无法获取、功能无法操作、与辅助技术不兼容等问题，使得视障群体在使用互联网过程中遇到多种障碍，面临“数字鸿沟”。为切实解决视障群体运用智能技术困难，提升 Windows 桌面应用无障碍化水平，特制定本指南。

解决的问题：当前 windows 系统无障碍可访问性接口调用方式复杂，造成第三方应用无障碍体验较差、视障群体无法顺畅使用各类桌面应用，尤以办公软件为甚。1、开发人员缺乏包容性设计理念、系统性的中文参考文档较少、没有可快速集成的代码示例；2、中文输入法的输入过程复杂、涉及适配环节多，未能有输入法的无障碍适配规范可供参考。

解决方案：搜狗输入法联合阳光读屏软件发布《Windows 应用无障碍可访问性开发指南》，包括 Windows 可访问性接口适配流程、技术文档、控件可访问性设计规则、中文输入法的无障碍设计规范，助力 Windows 应用快捷适配无障碍可访问性设计，助力 1700 万视障群体顺畅使用 Windows 应用、办公输入效率提升。

2 术语

2.1 基本概念

包容性设计：是一个设计过程（不限于界面或技术），其中针对具有特定需求的特定用户优化了产品，服务或环境。通常，该用户是极端用户，这意味着该用户有某些特定需求，而有时需要通过其他设计过程来监督。通过关注极端用户，融合设计将使他们能够使用它，

同时还将涵盖许多具有（临时）相似需求的用户。

信息无障碍：是指任何人（无论是健全人还是残疾人，无论是年轻人还是老年人）在任何情况下都能平等的、方便地、无障碍地获取信息、利用信息。

辅助技术：泛指运用科技的方法，或研发科技的装置，协助身心障碍者，重建或替代他们的某些能力或身体机能，改善他们的生活品质。而任何能够增加、维持、改进或促进使用者的个人能力的项目、设备或产品系统，则称为辅助工具或辅助功能。

2.2 无障碍设计原则

可感知性：信息和用户界面组件必须以用户可以感知的方式呈现给用户，用户必须能够感知所呈现的信息。

可操作性：用户界面组件和导航必须是可操作的，用户必须能够操作界面。

可理解性：信息和用户界面的操作必须是可理解的。用户必须能够理解信息以及用户界面的操作（内容，操作不能超出用户的理解范围）。

兼容性：内容必须足够健壮，可以被包括辅助技术在内的各种用户代理可靠地解释，随着技术和用户代理的发展，内容应该保持可访问性的稳定。

2.3 开发术语

读屏软件：又称为屏幕阅读器，是一种安装于电脑或手机上的应用程序，用来将文字、图形以及电脑界面的其他部分（用文本转语音技术）转换成语音或盲文。对于视障者或阅读障碍者甚有助益，有些人会搭配放大软件一起使用。

点显器：又名盲文显示器， 能够将计算机上的信息用盲文（点阵突起）同步显示，便于盲人摸读，通过与读屏软件的配合，能将读屏软件读出的文字通过盲文显示到点显器上。

语音合成：即文本到语音转换，是指文本合成为语音的技术。该技术用于与用户沟通时不能或不方便阅读屏幕的情况，这不仅开创了使用应用程序和信息的新方式，还能让那些无法阅读屏幕文本的人更轻松地了解和融入世界。

快捷键：又称为快速键、组合键或热键，指通过某些特定的按键、按键顺序或按键组合来完成一个操作，很多快捷键往往与 Ctrl 键、Shift 键、Alt 键、Fn 键以及 Windows 平台下的 Win 键和 Mac OS 系统上的 Cmd 鍵等配合使用。

控件：是一种图形用户界面元素，其显示的信息排列可由用户改变，例如视窗或文本框。控件定义的特点是为给定数据的直接操作（direct manipulation）提供单独的互动点。控件是一种基本的可视构件块，包含在应用程序中，控制着该程序处理的所有数据以及关于这些数据的交互操作。

焦点：计算机程序语言中所谓的焦点，就是关注的区域，当前光标被激活的位置，是哪个控件被选中，可以被操作。

浏览光标：读屏软件中的概念，读屏软件通过无障碍可访问性接口在程序内部虚拟出一种界面控件的呈现方式，用户通过快捷键在此虚拟界面中获取和操作信息，当快捷键浏览到一个虚拟对象时，我们就认为浏览光标聚焦到了这个对象。

3 辅助技术应用场景介绍

3.1 常用的辅助功能

视障群体因其视力受损的程度不同，存在视力较差、失明，及其他阻碍视觉信息传达的情况，在使用计算机过程中，不同视力障碍用户，所使用的辅助功能也不尽相同，下面我们将以四类视力障碍所适用的辅助技术来阐述不同视障群体使用计算机的方式。

3.1.1 色盲/色弱

色盲和色弱都是色觉异常的表现，色盲是指不能分辨颜色，其中最常见的是红色盲和绿色盲。色弱是指分辨颜色的能力差，其中最常见的是红色弱和绿色弱。但在视力障碍人群中，普遍存在色弱情况，如不能分辨浅色背景下的深色文本，或畏光等情况。目前，比较成熟的方案是，不同色弱/色盲情况采用不同的颜色方案，具体包括高对比度、颜色反转、加粗、颜色滤镜等（参考 Windows11-设置-辅助功能中的显示设置）。

3.1.2 低视力

双眼中较好眼视力小于 6/18（0.3），或视野半径小于 10°，但其仍能应用或有潜力应用视力去做或准备做各项工作，部分用户兼有色盲和色弱。此类用户需要借助放大镜来浏览屏幕内容，因存在色弱情况，放大镜中也会集成颜色方案调节功能。当视力远远无法看清文本细节，只能看清图片、控件轮廓时，就需借助读屏软件的朗读鼠标功能，或放大镜中的随选朗读功能。鼠标朗读，及通过鼠标移动，当鼠标停留在某控件、内容上时，程序会将该位置的信息通过语音朗读给用户，例如鼠标移动到网页链接时，程序会将“XXX 链接”朗读给用户。

3.1.3 失明

一级、二级视障用户是读屏软件最主要的用户群体，即仅通过键盘输入和操作信息，通过文本到语音转换接收信息。具体流程是，使用键盘快捷键浏览控件、屏幕内容，也可通过快捷键操作控件、屏幕内容，快捷键触发功能后，读屏软件应立即将操作的结果通过文本转语音技术反馈给用户，如操作失败或操作耗时较长，也应告知用户发生异常的原因。全盲用户也可使用点显器来代替或部分代替语音朗读功能，多维度获取信息，实现信息利用最大化。

3.1.4 视障兼有听障

海伦·凯勒就存在视障和听障两类障碍问题，此类用户必须借助触觉，也仅能通过触觉来获取信息，点显器是他们获取信息、传达信息最主要的方式。点显器除了可以显示盲文字符外，还可通过点显器上的 8 个字符键和其他功能键与计算机进行信息交换，实现内容的获取和操作。

3.2 常见的辅助接口

3.2.1 MSAA

Microsoft Active Accessibility 是一种基于组件对象模型 (COM) 的技术，可改进辅助功能与 Microsoft Windows 上运行的应用程序的工作方式。它提供了集成到操作系统中的动态链接库以及一个 COM 接口和 API 元素，这些元素提供了可靠的方法来公开有关 UI 元素的信息。

通过使用 Microsoft Active Accessibility 并遵循无障碍设计实践，开发人员可以使在 Windows 上运行的应用程序更容易被许多有视力、听力或运动障碍的人访问。一般来说，开发人员需要对 COM 对象和接口以及 Unicode 有一定程度的了解。

Windows XP 和 Windows Server 2003 中内置了对 Microsoft Active Accessibility 的完全支持。Microsoft Active Accessibility 也支持带有 Service Pack 6 (SP6) 及更高版本的 Windows NT 4.0 和 Windows 98。

参考： https://docs.microsoft.com/en-us/windows/win32/winauto/microsoft-active-accessibility

3.2.2 UIA

Microsoft UI Automation (UIA)是一个框架，通过提供对这些用户界面元素的编程访问，开发人员可以访问、识别和操作任何应用程序的 UI 元素。

参考：https://www.microfocus.com/documentation/silk-test/200/en/silktestworkbench-help-en/GUID-C11F8AD3-4E53-48CF-96C2-ABD817A9D8FC.html

3.2.3 IA2

IAccessible2 是 Microsoft Windows 应用程序的辅助功能 API。最初由 IBM 在代号 Project Missouri 下开发，IAccessible2 已被置于 Free Standards Group 的支持下，现在是 Linux Foundation 的一部分。IA2 填补了 MSAA 的空白，例如“支持文本控件、表格、超链接和可访问对象之间的关系”。它与 Linux（尤其是 Gnome）上的 Accessibility Toolkit (ATK) 相协调。IAccessible2 是对 MSAA 的补充，而不是替代品。

3.2.4 JAB

Java Access Bridge 是一种在 Microsoft Windows DLL 中公开 Java Accessibility API 的技术，使实现 Java Accessibility API 的 Java 应用程序和小程序对 Microsoft Windows 系统上的辅助技术可见。Java Accessibility API 是 Java Accessibility Utilities 的一部分，Java Accessibility Utilities 是一组实用程序类，可帮助辅助技术提供对实现 Java Accessibility API 的 GUI 工具包的访问。

参考：https://www.oracle.com/java/technologies/javase/javase-tech-access-bridge.html

4 实践概述

4.1.1 所有的应用都应考虑增加多种交互方式

一个良好的应用，应该支持多种交互方式，例如鼠标、键盘、触控手势以及其它辅助性交互方法。具有多种交互方式的应用，既可以满足特殊人群用户的需求，也可以满足普通人在特定环境下的使用需求，从而扩大用户范围，使应用 UI 交互更加人性化。

要实现多种交互方式，应用软件的开发者不必从零开始，自己制造轮子，几乎所有的成熟操作系统都提供了相应的可访问性接口，或称辅助技术接口。只要应用软件开发者严格遵守接口的规范，完整的实现各项方法，准确填充可访问性属性值，就可以开发出一个良好的具有多种交互方式的应用软件。

我国即将进入老龄化社会，老年人和残障用户在软硬件交互中遇到的问题会越来越突出。对于开发者，怎样进一步提升用户体验，满足弱势群体的使用需求，成为新一代软件从业者需要共同努力解决的问题。

随着我国无障碍相关法律法规的完善，残障人群和老年人群的维权意识也在逐步提升，改进应用的无障碍体验，不但可以体现软件设计者的人文关怀，同时也提前规避潜在的法律风险。

4.1.2 提前规划

大量的产品无障碍问题案例中，一般情况下应用软件的功能已经较为成熟，无障碍方面的问题才逐步暴露出来。

这种情况下，如果要改进软件的无障碍体验，可能需要修改多处界面交互代码，对于某些大型软件项目要实现所有界面的适配是一项繁重的工作。

因此，我们提倡在软件初期设计阶段就应该将无障碍的相关需求考虑进来，将 Windows 可访问性接口的实现和界面库进行结合。让界面的每一个基础元素具备无障碍属性信息，避免项目后期需要付出大量的人力去做 UI 的底层重构。

4.1.3 开发建议

要使 Windows 应用软件具有良好的可访问性，开发者并不一定需要完全自主实现可访问性接口的底层代码。这对于大部分情况并无必要。

较为推荐的做法是在界面开发中，使用已知的具有良好可访问性设计的 UI 库。除非确实具有自主开发 UI 框架的必要，例如已有 UI 库在某些方面无法满足需求，或其它原因，才需要完全从头开发界面框架。这种情况下，需要在框架接口设计中，参考下文，实现 Windows 可访问性接口，以及发送恰当的界面变化事件。

下面列出一些推荐的 UI 框架：

名称	支持的语言
WPF（Windows Presentation Foundation）	C++/C#
MFC（Microsoft Foundation Classes）	C++
wxWidget	C++/Python/JavaScript ...
QT	C++

4.2 属性和导航

4.2.1 属性概述

目前 Windows 平台常见的可访问性技术接口包括 MSAA、UIA、IA2 和 JAB 等。

尽管这些可访问性接口的具体实现不尽相同，但交互逻辑均类似。

Windows 应用通过这些可访问性接口，将视觉元素信息转换为具有一定格式的文本信息，传递给辅助技术软件（屏幕阅读器等），让用户不完全依赖显示器也可以使用应用软件所提供的功能。

这里所说的用来替代视觉元素信息的各种数据结构，统称为“可访问性属性值”。

最基础的可访问性属性值有 Name(名称)，role（角色），state（状态），location（屏幕坐标）。

这些属性值有的是以宏和常量的方式提供，还有的是字符串。

作为填充这些属性的应用开发者，应尽可能准确、恰当的填充其值。既要避免属性的缺失，也要避免属性信息的冗余和值的混用。

属性值的缺失，会导致屏幕阅读器用户无法理解元素的作用，无法和其它元素进行区分，甚至无法执行相应的操作。

信息冗余和值的混用，也会带来不良的用户体验，冗余信息不但影响信息传递的效率，而且过于基础、过于常识性的提示会给用户带来理解力低下的心理暗示，会让用户感到被冒犯；

值的混用，也是一个比较常见的问题，屏幕阅读器等辅助技术软件往往依赖元素的特定属性进行有针对性的提示和操作，例如根据元素的 Role 属性信息确定控件类型，根据 State 属性信息确定控件的状态。

4.2.2 Name 属性

Name 一般代表的是屏幕元素的名称，如果一个控件有文本标签，例如“确定”、“取消”，那么这些可见的文本作为 Name 属性非常恰当；如果一个元素是用图标表示其含义，例如代表关闭的“X”形图标，代表设置的“扳手或齿轮”图标，Name 属性可以用图标所代指的含义来

填充，例如“关闭”、“设置”等。

Name 属性一般使用字符串类型的数据，这种情况下，如何提供恰当的文本是开发者需要考虑的问题。

如果一个元素有与之对应的文本标签，例如一个文本框前面有“姓名”的文本提示，那么用“姓名”作为该文本框的 Name 属性值非常合适。

如果一个元素在视觉上是用图标表示，开发者应将图标所代指的含义，或者说该元素的功能，用文本填充到 Name 属性。

比较常见的有用扳手或齿轮图标表示的“设置”，用加号表示的“更多”，用箭头表示的“展开”。

填充字符串形式的可访问属性值，要使文本尽可能的准确而精炼，尽量避免冗余和歧义，这一点非常关键。

如果说一个 UI 设计布局是否合理，外观是否美观，对于普通用户的软件体验非常重要，那么可访问性属性的层次组织关系是否清晰，属性文本是否准确明了，则严重影响屏幕阅读器用户的交互体验。

如果软件有多语言需求，这些填充的文本还要考虑本地化转换问题，这不是本文的重点，此处不做赘述。

在实际开发中，比较常见的关于 Name 属性的问题除了以上这些之外，还有的 UI 框架，会将图标的原文件名作为可访问性属性值填充进来。

虽然提供了属性值，但是往往是不准确的，这取决于美工人员提供的素材，命名是否规范，每个图标的名称是否具有实际的含义。

对于程序人员，不依赖资源文件的命名，主动填充恰当的可访问属性值，显然是更稳妥负责的做法。

下面代码片段以 MSAA 接口为例，展示应用软件如何通过可访问性接口填充 Name 属性信息：

```
IFACEMETHODIMP AccServer::get_accName(
    VARIANT varChild,
    BSTR *pszName)

{
    *pszName = NULL;
    if (!m_controlIsAlive)
    {
        return RPC_E_DISCONNECTED;
    }

    if ((varChild.vt != VT_I4) || (varChild.lVal > m_pControl->GetCount()))
    {
        *pszName = NULL;
        return E_INVALIDARG;
    }

    *pszName = SysAllocString(L"关闭");
    if (!pszName)
    {
        return E_OUTOFMEMORY;
    }
    return S_OK;
}
```

4.2.3 Role 属性

Role 属性代表了某界面元素的类型，例如 Button（按钮）、StaticText（静态文本）、CheckBox（复选框）、ComboBox（组合框）等等。

这些其实就是 UI 设计中的控件类型。每个类型都有与之对应的交互方法，例如单击选中、取消选中，单击按下按钮等。

屏幕阅读器需要正确识别控件的类型，才能使用正确的 API 对其进行专项支持。

因此应用软件需要根据界面元素的功能，对控件进行分类，选择最恰当的 Role 值进行填充。

Role 值一般接口以宏定义和常量的方式提供，大部分可访问性接口会根据系统语言设定，对其进行本地化转换，省去了开发者维护的成本，也提高了跨应用的体验一致性。

下面列出部分 MSAA 接口中，关于 Role 属性的定义，以供参考：

常量	描述
ROLE_SYSTEM_CHECKBUTTON	对象表示一个复选框：选中或取消选中不依赖其他选项
ROLE_SYSTEM_COMBOBOX	对象表示一个组合框：由编辑框控件和与之关联的列表组成，提供一个预定义选项的集合。
ROLE_SYSTEM_LINK	对象表示了一个到其他内容的链接：这个对象或许看起来相文本或图片，但是其行为类似按钮。
ROLE_SYSTEM_LIST	对象表示一个列表框，允许用户选择一个或多个项目。

完整定义请见 https://docs.microsoft.com/en-us/windows/win32/winauto/object-roles

下面代码片段以 MSAA 接口为例，展示应用软件如何通过可访问性接口填充 Role 属性信息：

```
IFACEMETHODIMP AccServer::get_accRole(
    VARIANT varChild,
    VARIANT *pvarRole)
{
    pvarRole->vt = VT_EMPTY;
    if (!m_controlIsAlive)
    {
        return RPC_E_DISCONNECTED;
    }

    if ((varChild.vt != VT_I4) || (varChild.lVal > m_pControl->GetCount()))
    {
```

```
            pvarRole->vt = VT_EMPTY;
            return E_INVALIDARG;
        }
        pvarRole->vt = VT_I4;
        if (varChild.lVal == CHILDID_SELF)
        {
            pvarRole->lVal = ROLE_SYSTEM_LIST;
        }
        else
        {
            pvarRole->lVal = ROLE_SYSTEM_LISTITEM;
        }
        return S_OK;
    }
```

4.2.4 State 属性

State 属性反应了一个屏幕元素的状态信息，例如是否聚焦、是否选中、是否可见、是否可用等等。

这些信息从视觉的角度一般多用不同的颜色加以区分，选中的元素通常用高亮边框、聚焦元素通常用反色表示、不可用通常用灰色表示。

对于屏幕阅读器用户，获取正确的 State 属性信息，是区分屏幕元素状态的唯一途径。

如果一个按钮是灰色状态，而用户无法感知的话，点击操作之后就会发现软件没有任何的反应。这种情况，会对屏幕阅读器用户造成困扰，因为根本不知道发生了什么，可能是操作的结果没有提示，也可能是操作根本就没有发生，这种体验对于视障用户而言，显然是非常糟糕的。

还有一些元素，包含在某个功能下，需要展开才能进行交互，由于 UI 交互和可访问性接口的实现是两套逻辑，尚未显示的元素是可能提前被屏幕阅读器获取到的。这种情况下，如果元素给出了不可见的状态，用户就可以很清楚的了解，这些元素目前是隐藏的。同样的，如果没有给出正确的提示，用户很可能操作之后得不到任何回馈，甚至会出现本来听到是张

三、点击却变成李四的尴尬局面。

当然，我们更推荐，在设计可访问性接口实现的同时，应尽量使其提供的信息和界面元素相匹配。如果某个元素不可见，最佳的做法是在可访问性导航序列中移除该元素。这样既减少了可访问性信息的冗余，又保持了两套交互逻辑的一致性。

下面列出部分 MSAA 接口中关于 State 属性的定义，以供参考：

常量	描述
STATE_SYSTEM_FOCUSABLE	对象处于激活的窗口中，并且已准备好接受焦点。
STATE_SYSTEM_FOCUSED	对象具有焦点。不要混淆聚焦和选中。
STATE_SYSTEM_BUSY	控件此时无法接受输入。
STATE_SYSTEM_INVISIBLE	对象以程序方式隐藏。例如用户激活菜单之前菜单项目处于隐藏状态
STATE_SYSTEM_OFFSCREEN	对象被裁剪或者滚动到了试图之外，但这不属于程序隐藏。如果用户增大视口，则对象将在计算机屏幕上可见。

完整定义请见 https://docs.microsoft.com/en-us/windows/win32/winauto/object-state-constants

下面代码片段以 MSAA 接口为例，展示应用软件如何通过可访问性接口填充 State 属性信息：

```
IFACEMETHODIMP AccServer::get_accState(
    VARIANT varChild,
    VARIANT *pvarState)
{
    pvarState->vt = VT_EMPTY;
    if (!m_controlIsAlive)
    {
        return RPC_E_DISCONNECTED;
    }

    if ((varChild.vt != VT_I4) || (varChild.lVal > m_pControl->GetCount()))
    {
        pvarState->vt = VT_EMPTY;
        return E_INVALIDARG;
    }
    if (varChild.lVal == CHILDID_SELF)
    {
        return m_pStdAccessibleObject->get_accState(varChild,  pvarState);
    }
    else  // 列表项的处理
    {
        DWORD flags = STATE_SYSTEM_SELECTABLE |
STATE_SYSTEM_FOCUSABLE;
        int index = static_cast<int>(varChild.lVal - 1);
        if (index == m_pControl->GetSelectedIndex())
        {
            flags |= STATE_SYSTEM_SELECTED;
            if (GetFocus() == m_hwnd)
            {
                flags |= STATE_SYSTEM_FOCUSED;
            }
        }
        pvarState->vt = VT_I4;
        pvarState->lVal = flags;
    }
    return S_OK;
}
```

4.2.5 Location 属性

Location 表示了一个界面元素的位置信息。如果对象的形状不像矩形，则应返回包含整

个对象的最小矩形的坐标范围。

屏幕阅读器用户大部分交互操作不依赖屏幕显示，但是这不意味着位置信息没有作用。

就像每一个控件都应该有其类型一样，每个控件当放置在界面上时也必然有一个位置信息。

屏幕阅读器可以利用元素的位置将鼠标指针放置到某元素，从而实现鼠标单击或右击。这对于某些未完全实现可访问性接口的应用尤为重要，它给屏幕阅读器用户操作该软件留下了一线可能。

除此之外，屏幕阅读器根据元素的位置信息，还可以对可访问性信息进行位置排序，实现近似方位上的导航浏览。

下面代码片段以 MSAA 接口为例，展示应用软件如何通过可访问性接口填充 Location 属性信息：

```
IFACEMETHODIMP AccServer::accLocation(
    long *pxLeft,
    long *pyTop,
    long *pcxWidth,
    long *pcyHeight,
    VARIANT varChild)
{
    *pxLeft = 0;
    *pyTop = 0;
    *pcxWidth = 0;
    *pcyHeight = 0;
    if ((varChild.vt != VT_I4) || (varChild.lVal > m_pControl->GetCount()))
    {
        return E_INVALIDARG;
    }
    if (!m_controlIsAlive)
    {
        return RPC_E_DISCONNECTED;
    }
```

```
        if (varChild.lVal == CHILDID_SELF)
        {
            return m_pStdAccessibleObject->accLocation(pxLeft, pyTop, pcxWidth,
pcyHeight, varChild);
        }
        else
        {
            RECT rect;
            if (m_pControl->GetItemScreenRect(varChild.lVal - 1, &rect) == FALSE)
            {
                return E_INVALIDARG;
            }
            else
            {
                *pxLeft = rect.left;
                *pyTop = rect.top;
                *pcxWidth = rect.right - rect.left;
                *pcyHeight = rect.bottom - rect.top;
                return S_OK;
            }
        }
    }
```

4.2.6 导航概述

应用软件提供了足够的可访问性属性信息，对于辅助技术用户而言，如何在信息之间高效的导航，充分利用获取到的这些信息呢？

Windows 平台，可访问性接口提供的导航方式大致可以分两类，焦点导航和层级导航。

以此为基础，许多辅助技术软件发展出了各具特色的其他导航方式，例如方位导航、类别导航等等。

其目的都是提高非视觉的交互效率，简化操作流程，缩小辅助技术用户和普通视觉操作用户的差距。

4.2.7 焦点事件

焦点导航，按下快捷键后，可以将焦点从一个元素移动到另一个元素。常用的焦点导航快捷键有 Tab、方向键、回车、退格等。

在焦点移动过程中，前一个元素失去焦点，后一个元素获得焦点，同时伴随着焦点变化事件。

应用软件开发者需保证上述提到的快捷键可以在自定义控件之间移动，同时还应调用对应的事件函数发送焦点变化事件。

辅助技术开发者在事件回调函数中捕获焦点事件，调用相应的 API 从事件中构建可访问性对象获取所需可访问性属性信息。

下面代码片段以 MSAA 接口为例，展示应用软件如何通过可访问性接口发送焦点变化事件信息：

```
void CustomListControl::SelectItem(int index)
{
    m_selectedIndex = index;
    if (m_selectedIndex >= static_cast<int>(m_itemCollection.size()))
    {
        m_selectedIndex = static_cast<int>(m_itemCollection.size()) - 1;
    }

    // 发送 WinEvents 事件
    NotifyWinEvent(EVENT_OBJECT_SELECTION , m_controlHwnd , OBJID_CLIENT, m_selectedIndex + 1);
    if (GetIsFocused())
    {
        NotifyWinEvent(EVENT_OBJECT_FOCUS , m_controlHwnd , OBJID_CLIENT, m_selectedIndex + 1);
    }
```

```
    // 强制刷新界面
    InvalidateRect(m_controlHwnd,  NULL,  TRUE);
}
```

4.2.8 层级导航

界面元素在布局上存在层次关系，窗口的不同区域有不同的功能划分，不同的功能区域，又可以由多个控件组合实现。

对于可访问性接口而言，一般的原始数据是树状的，树上的每个节点，包含了若干可访问性属性信息，称作可访问性对象。每一个可访问性对象对应于界面上的一个可见或不可见的元素。对象之间存在兄弟和父子关系。通过接口提供的 API 在节点之间进行遍历，称为层级导航。

可访问对象的这种层次结构也可以体现界面元素在逻辑上的从属关系，例如列表内包含了列表项目，表格内包含了单元格。

当然这种层级关系是非常灵活的，很多不同类型的可访问性对象都可以互相包含，也可以多层嵌套，构建出非常复杂的树状结构。

对于辅助技术和辅助技术用户而言，希望这棵树是枝繁叶茂而又层次清晰的，不希望有无效的节点。因为这些不包含有效可访问性属性的节点，不仅给辅助技术软件开发者处理冗余数据带来麻烦，对于使用者来说，偶尔夹杂获取到的无用信息也让用户十分苦恼。

下面代码片段以 MSAA 接口为例，展示应用软件如何通过可访问性接口为辅助技术提供在本应用内实现层级导航的能力：

```
// 获取父对象
//
```

```
IFACEMETHODIMP AccServer::get_accParent(
    IDispatch **ppdispParent)
{
    *ppdispParent = NULL;
    if (!m_controlIsAlive)
    {
        return RPC_E_DISCONNECTED;
    }

    return m_pStdAccessibleObject->get_accParent(ppdispParent);
}

// 获取子对象的数量
//
IFACEMETHODIMP AccServer::get_accChildCount(
    long *pcountChildren)
{
    *pcountChildren = 0;
    if (!m_controlIsAlive)
    {
        return RPC_E_DISCONNECTED;
    }

    *pcountChildren = m_pControl->GetCount();
    return S_OK;
}


// 获取子对象。 因为控件中的列表项目是元素,
// 不是对象，返回 S_FALSE.
//
IFACEMETHODIMP AccServer::get_accChild(
    VARIANT varChild,
    IDispatch **ppdispChild)
{
    *ppdispChild = NULL;
    if (!m_controlIsAlive)
    {
        return RPC_E_DISCONNECTED;
    }

    if ((varChild.vt != VT_I4) || (varChild.lVal > m_pControl->GetCount()))
    {
        return E_INVALIDARG;
```

```
        }
        return S_FALSE;
    }
```

5、搜狗拼音输入法无障碍实践

5.1. 背景

初期，此文档是搜狗拼音 PC 端公益项目中的一个读屏需求的技术方案。意图使搜狗输入法内带的 win32 应用被读屏软件广泛支持。

后来在与 NVDA、阳光读屏、争渡读屏等开发社区交流时获悉，目前市面上 win32 应用的“无障碍”情况并不乐观。问题主要集中在使用小众界面库开发的软件，在设计之初未考虑无障碍接口。另外一小部分，虽然实现了无障碍接口但实现并不规范，导致读屏软件无法正确“读取软件界面”。

本章将以 DuiLib 界面库上实现 IAccessible 接口为例，以 Win10 系统自带的讲述人读屏方式为标准，为 Windows 应用开发者提供一个详尽的无障碍开发指南。

5.2 应该选择哪个辅助接口

5.2.1 为什么需要辅助接口

Win32 应用开发者不可能在开发应用时同时也开发与之配套的无障碍辅助软件，反之亦然。因此**需要有一个中间层去连接 Win32 应用与辅助功能软件**，使双方可以按各自的职能开发产品。最终无需互相适配对方，为有障碍人士提供正常的使用体验。

换种说法，软件的用户交互可抽象为**“呈现（Output）”、“操作（Input）”**两类。辅助接口的职能是将这两部分标准化、数据化，辅助软件拿到标准“呈现数据”后，对不同的障碍人群使用不同的呈现方式。如对视觉障碍人士使用声音（读屏）或触觉（盲文摸读）来呈现数

据。

对于“操作”也是一样，辅助软件拿到规范化的“操作数据”后，可以使用声控、眼动等手段实现视障和肢体障碍的操作能力。

5.3 辅助接口的几个关键点

辅助接口“呈现（Output）”、“操作（Input）”的数据化，主要通过如下几个方面：

5.3.1 呈现-控件类型

指控件按其行为不同而区分的类型，用户可通过控件类型来判断当前的操作大致是个什么行为。

如编辑框（EditBox）是用来输入文字，按钮（Button）是用来执行一个动作；类似还有单选（RadioButton）按钮、复选（CheckButton）按钮、菜单（Menu）、组合框（CombBox）、列表（Listbox）项等。

5.3.2 呈现-控件名称

控件名称是用来描述，当前控件在软件中对应的具体功能是什么。如单选按钮或复选按钮上的文本。

5.3.3 呈现-控件状态

状态主要主要是指当前控件是否拥有焦点、是否被选中等状态。

如“单选按钮”、“复选按钮”有“选中/未选中”两种状态。

5.3.4 呈现-控件取值

部分类型的控件在使用过程中，可以被用户行为更改为不同的内容。这些复杂（对应简单的选中状态）的控件数据，在辅助接口种统一用“控件取值”来决定。

如文本编辑框（EditBox）可以被用户输入任意内容。类似还有组合框（CombBox）中多个选项的具体选中值、某些操作进度条的实时进度等。

5.3.5 交互-焦点控件

软件界面交互流程，对于鼠标无碍用户来说，可以是移动鼠标到一个控件并执行点击。

在这一系列的步骤里，软件交互的首先是搞清楚“用户要操作哪个控件”，然后是执行该控件的点击响应函数。

而将其标准化的办法是，将鼠标点击为为 2 步：①获取屏幕坐标点对应的控件对像；②执行该对象的默认响应函数。如此，应用开发者只需实现“IAccessible:: accHitTest” 接口和每一个控件默认动作响应即可完成软件交互。具体是用何种方式来移动焦点控件或发送执行默认动作的命令，不同的辅助软件可选用不同的方式。

5.3.6 交互-补充说明

除了焦点控件的相关逻辑，用户交互中还有关闭/最小化窗口、切换活动窗口等交互方式。

5.4 IAccessible 接口

5.4.1 为什么搜狗拼音选用 IAccessible

适用广度：IAccessible 接口从最早的 XP 系统到现在的 WIN11 系统，都已被微软很好的

支持，且各种辅助软件也都做了适配。

适配成本：IAccessible 相比其它，其接口更简单、逻辑清晰。虽然简单，但它覆盖了无障碍辅助软件 90%以上的诉求。所以，无论是无障碍入门开发，还是大型 Win32 商业软件的无障碍适配，在不考虑自动化测试需求的前提下，都推荐使用 IAccessible 接口来做。

产品需求：因搜狗输入法 PC 版只需完成鼠标、键盘输入的无障碍输入即可，IAccessible 接口正好够用。

5.4.2 辅助软件与已 IAccessible 的交互原理

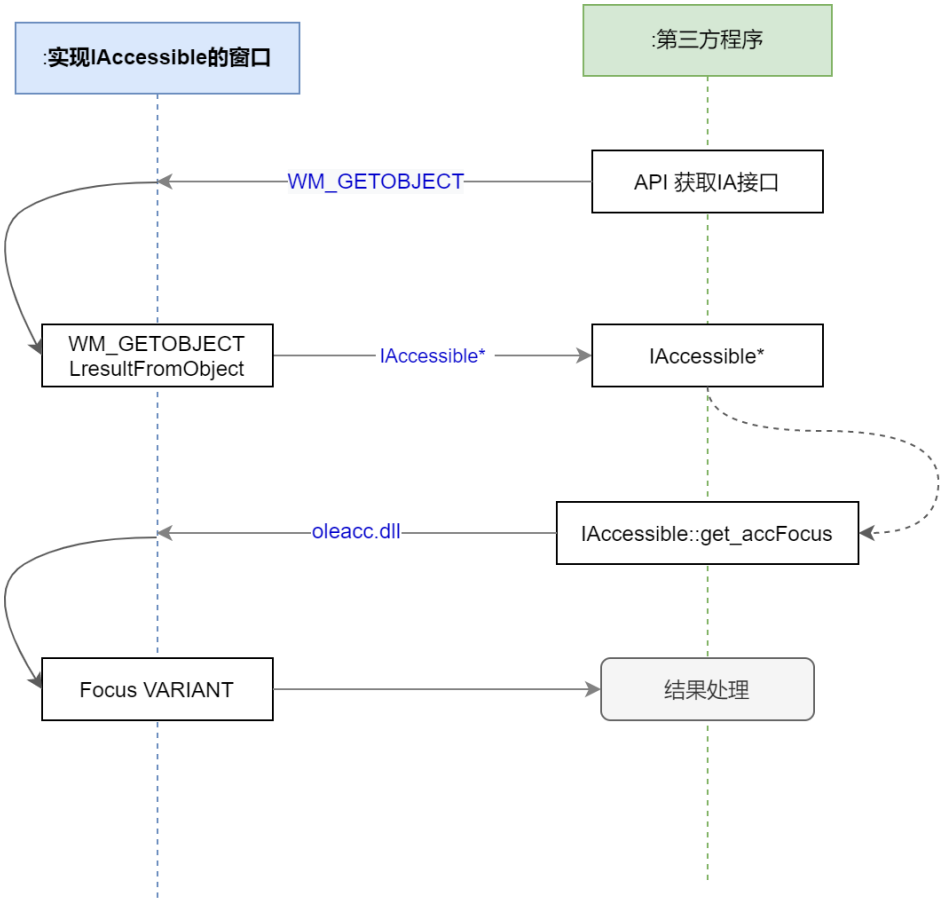
辅助程序获窗口界面元素的一般流程为：

使用 WM_GETOBJECT 获取界面根节点接口(IAccessiable)

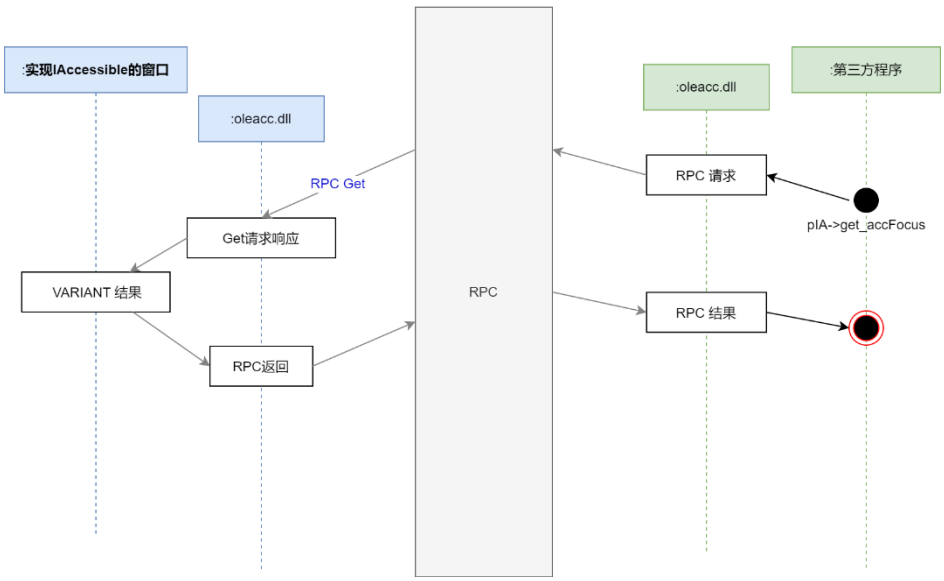
使用 IAccessiable::get_accFocus 获取当前焦点控件

使用 IAccessiable::get_accName/getDesc/GetValue/GetRole()等接口获取节点详细信息

使用 IAccessiable::get_accChildCount() 及 IAccessiable::get_accChild() 获取子节点(IAccessiable)列表



5.4.3 IAccessible 的实际通信流程



5.5 为 DuiLib 增加通用的辅助接口

5.5.1 关键路径

如前描述，辅助软件是通过 Com 方式获取 IAccessible 接口，然后使用接口函数获取界面对象的数据。所以在 DuiLib 实现 IAccessible 接口大致有四个部分的工作。

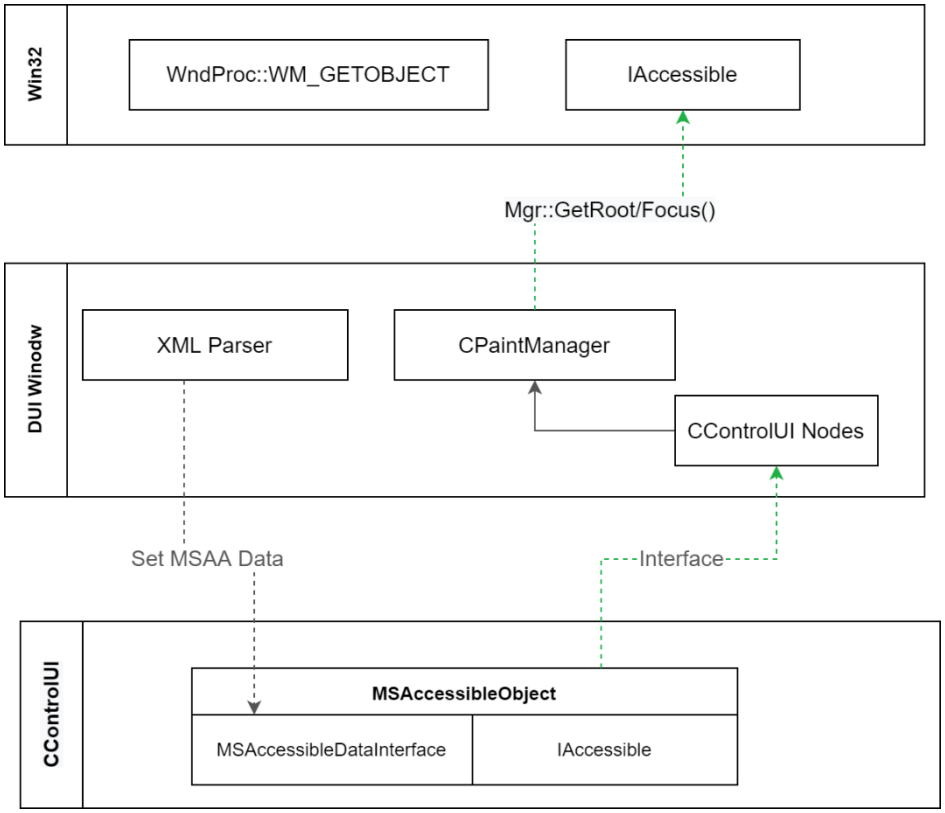
①接口：实现一个从 interface IAccessible 继承的类，用于以 Com 方式返回对象到辅助程序中

②数据：为 CControlUI 增加 IAccessible 所需的数据接口，用于设置或返回辅助程序必须的 IAccessible 对象的数据

③响应：为 WindowImplBase::HandleMessage()增加 WM_GETOBJECT 消息的处理，以响应辅助软件获取 IAccessible 对象的请求

④通知：当软件界面发生改变时，通知辅助软件当前发生的改变

大致的架构设计如下：



5.5.2 接口

实现 IAccessible 接口的几个重要函数

// 呈现-控件名称

STDMETHODIMP get_accName(VARIANT varChild， __RPC__deref_out_opt BSTR* pszName);

// 呈现-控件取值

STDMETHODIMP get_accValue(VARIANT varChild， __RPC__deref_out_opt BSTR* pszValue);

// 呈现-控件类型

STDMETHODIMP get_accRole(VARIANT varChild， __RPC__out VARIANT* pvarRole);

// 呈现-控件状态

STDMETHODIMP get_accState(VARIANT varChild， __RPC__out VARIANT* pvarState);

// 交互-当前焦点控件

STDMETHODIMP get_accFocus(__RPC__out VARIANT* pvarChild);

5.5.3 数据

为 ControlUI 新增 4 个属性，并支持在 XML 中定义

序号	名称	功能	内部设置接口
1	msaa_desc	IAccessible::get_accDescription 的返回数据 设置控件的描述文本，用于读屏软件的朗读文本提供	::set_msaa_desc_text(...)
2	msaa_text	IAccessible::get_accValue 的返回数据 用于说明 Combox/ProgressBar 等控件的文本描述值	::set_msaa_text(...)
3	msaa_name_	IAccessible::get_accName 的返回数据 用于说明控件的名称，一般为软件内部使用	::set_msaa_name(...)
4	msaa_shortkey	IAccessible::get_accKeyboardShortcut 的返回数据 用于描述当前控件的默认动作执行热键	::get_msaa_shortkey(...)

```
<Button name="minbtn" padding="0，6，4，0"
        width="30" height="30"
        normalimage="file='img\mini_normal.svg' corner='3，3，3，3'"
        hotimage="file='img\mini_hover.svg' corner='3，3，3，3'"
        pushedimage="file='img\mini_click.svg' corner='3，3，3，3'"
      msaa_desc="最小化窗口"
      msaa_text="最小化"
      msaa_shortkey="Ctrl+Shift+N"
       />
```

ControlUI 中新增两个成员变量，用于返回控件类型和状态

序号	名称	功能
1	msaa_status_	如何为 IAccessible::get_accState()返回数据 取值说明: 使用 OleAcc.h 中的 STATE_SYSTEM_*宏定义，每个控件的多种状态可以使用 ROLE_SYSTEM_*组合得出 参加: Object State Constants (Oleacc.h) - Win32 apps | Microsoft Docs
2	msaa_role_	为 IAccessible::get_accRole()返回数据 取值说明: 使用 OleAcc.h 中的 ROLE_SYSTEM_*宏定义，每个控件只有一个 Role 参见: Object Roles (Oleacc.h) - Win32 apps | Microsoft Docs

5.5.4 响应

在 CPaintManagerUI 中实现 WM_GETOBJECT 的处理

有此实现后，可以实现以下几种需求:

辅助软件，可通过 AccessibleObjectFromWindow(…)获取窗口的根节点 IAccessible 对象

辅助软件，可通过 IAccessible::accHitTest()获取鼠标所在位置指向的 IAccessible 对象

辅助软件，可通过 IAccessible::get_accFocus()或 AccessibleObjectFromWindow()获取当前拥有焦点的 IAccessible 对象

辅助软件，可通过 AccessibleObjectFromWindow 获取应用发送的特殊 objectID 对应的 IAccessible 对象

```
LRESULT CPaintManagerUI::OnGetObject(WPARAM wParam,  LPARAM lParam,  BOOL& bHandled) {
    if (!uicontrol_msaaex_itf_) {
        bHandled = false;
        return 0;
    }
```

```
    if ((LONG) lParam == OBJID_CLIENT) {
        LRESULT lRes = ::LresultFromObject(IID_IUnknown, wParam,
static_cast<LPUNKNOWN>(uicontrol_msaaex_itf_));
        bHandled = true;
        return lRes;
    } else if ((LONG) lParam >= MSAA_OBJID_START && (LONG) lParam <
MSAA_OBJID_END) {
        VARIANT var;
        if (this->find_add_add_object(get_top_node(), &var) == S_OK) {
            LRESULT lRes = ::LresultFromObject(IID_IUnknown, wParam,
var.pdispVal);
            bHandled = true;
            return lRes;
        }
    }
    bHandled = false;
    return 0;
  }
```

实现一个 IAccessible 的派生类

在辅助软件需要时，返回 IAccessible 对象的实际数据。

以下代码演示：返回当前对象的子对象

```
STDMETHODIMP UIControlMSAAEx::get_accChild(VARIANT varChild ,
__RPC__deref_out_opt IDispatch** ppdispChild) {
    auto ptr = dynamic_cast<CContainerUI*>(current_node_ ?
current_node_ : root_node_);
    // 可理解为 varChild.lVal 从 1 开始，因为 0 表示自己
    auto pchild = ptr->GetItemAt(varChild.lVal - 1);
    auto newobj = alloc_msaa_object(root_node_, pchild);
    *ppdispChild = static_cast<IDispatch*>(newobj);
    return S_OK;
}
```

以下代码演示：返回当前拥有焦点的对象。

```
STDMETHODIMP UIControlMSAAEx::get_accFocus(VARIANT* pvarChild) {
    auto focus = root_node_->GetManager()->GetFocus();
    auto ptr = alloc_msaa_object(root_node_, focus);
    pvarChild->vt = VT_DISPATCH;
    pvarChild->pdispVal = dynamic_cast<IDispatch*>(ptr);
    return S_OK;
}
```

以下代码演示：返回当前对象的 Name。

Name/Value/Description/Role/State/KeyboardShortcut 等数据内容，均可按如下方式提供:

```
STDMETHODIMP UIControlMSAAEx::get_accName(VARIANT varChild ,
BSTR* pszName) {
    *pszName = ::SysAllocString(child->get_msaa_text());
    return S_OK;
}
```

以下代码演示：返回当前对象的屏幕位置。

如果该位置返回正确，Windows 系统自带的讲述人（其它读屏软件无此行为）会按照这个矩形绘制一个提示框。

```
STDMETHODIMP UIControlMSAAEx::accLocation(long* pxLeft, long* pyTop,
long* pcxWidth, long* pcyHeight, VARIANT varChild) {
    CControlUI* child_control = pmgr->FindSubControlById(root_node_ ,
varChild.lVal);
    RECT child_rect = child_control->GetPos();
    POINT top_left{child_rect.left, child_rect.top};
    ::ClientToScreen(pmgr->GetPaintWindow(), &top_left);
    *pxLeft = top_left.x;
    *pyTop = top_left.y;
    *pcxWidth = child_rect.right - child_rect.left;
    *pcyHeight = child_rect.bottom - child_rect.top;
    return S_OK;
}
```

以下代码演示：返回指定坐标处的 IAccessible 对象。

如果该接口返回正确，鼠标在界面内 Hover 控件时，读屏软件会朗读 Hover 控件。

```
STDMETHODIMP UIControlMSAAEx::accHitTest(long xLeft , long yTop ,
VARIANT* pvarChild) {
    CPaintManagerUI* pmgr = root_node_->GetManager();
    POINT pt{xLeft, yTop};
    ::ScreenToClient(pmgr->GetPaintWindow(), &pt);
    CControlUI* child_control = pmgr->FindFocusControlByPoint(nullptr, pt);
    pvarChild->vt = VT_I4;
    pvarChild->lVal = pmgr->FindOrAddControlId(child_control);
    return S_OK;
}
```

5.5.5 通知

必要时，通知辅助软件更新

	应用中的动作	可能的用户操作方式	通知辅助软件的方法	注意事项
1	焦点控件改变	Tab 选择下一个焦点控件 ↑↓←↓切换接点控件 鼠标选择控件	LONG id = GetControlId(m_pFocus); NotifyWinEvent(EVENT_OBJECT_FOCUS, m_hWnd, id, CHILDID_SELF);	①注意 clientID 各不重复，因为某些辅助软件（NVDA）会使用缓存机制存储对应 ID 的辅助数据，这样可以降低重复的接口获取操作，提升效率； ②焦点控件改变时，辅助软件获取新的焦点对象后，会使用 get_accState 获取对象的状态，该状态必须包含 STATE_SYSTEM_FOCUSED 标志位。因为部分读屏软件不朗读没有 STATE_SYSTEM_FOCUSED 标志位的对象

2	控制状态改变	空格操作"单/复选按钮" 快捷键触发"展开快捷菜单等" 鼠标点选"单/复选按钮"、展开菜单、选择Combox、选择列表等	NotifyWinEvent(EVENT_OBJECT_STATECHANGE, m_hWnd, id, CHILDID_SELF); NotifyWinEvent(EVENT_OBJECT_NAMECHANGE, m_hWnd, id, CHILDID_SELF); NotifyWinEvent(EVENT_OBJECT_FOCUS, m_hWnd, id, CHILDID_SELF);	
3	进度提示	进度条进度发生改变（自动行为，与用户无关）	NotifyWinEvent(EVENT_OBJECT_VALUECHANGE, m_hWnd, id, CHILDID_SELF);	

5.6 I 焦点控件的控制及显示

5.1. 按键逻辑

Tab 键逻辑：包括 Tab/Shift+Tab，在原 DuiLib 方案基础上，增加用户操作强制刷新 NextTab 的能力。可参考：CPaintManagerUI::SetNextTabControl 的实现

上下左右键：应在菜单、列表、下拉框中实现方向键的响应

Space/Enter 键：执行焦点控件的默认动作即可

5.2. 状态显示：

可使用 m_bFocused 判断当前控件是否焦点状态

焦点态控件可使用 DrawFocusRect()绘制边框

5.7 辅助软件获取窗口内容的伪代码

```
// 窗口句柄
HWND hWnd = FindWindow(find_clsname,  find_winname);

// 获取 IA 接口
IAccessible* pIAcsb = nullptr;
HRESULT hr = AccessibleObjectFromWindow(hWnd , OBJID_CLIENT ,
IID_IAccessible,  (void**)&pIAcsb);

// 获取控件信息
hr = pIAcsb->get_accName(var2,  &bstrName);
hr = pIAcsb->get_accDescription(varID,  &bstrName);
hr = pIAcsb->get_accRole(varID,  &varOut);

// 获取焦点控件
hr = pIAcsb->get_accFocus(&varOut);
// 获取焦点控件的 IA 接口
varOut.pdispVal->QueryInterface(IID_IAccessible,  (void**)&pIASub));

// 获取内容
pIASub->get_accName(var2,  &bstrName);
```

5.8 开发及测试工具

AccExplorer32.exe：枚举窗口 Acc 控件树

AccEvent.exe：监视当前的 msaa 对象更改事件

系统讲述人：系统自带的屏幕朗读应用。其对辅助接口的判断要求最严格、规范。所以适配讲述人后，其它读屏软件基本都可正常工作。

6 如何获取 Windows 输入法候选列表

6.1 概述

本章讲述在 windows 操作系统中如何通过系统提供的输入法接口获取当前输入法的候选列表信息。在全屏游戏、或需要自绘输入法候选列表、或无障碍辅助软件中均需使用此技术。

在阅读之前，请务必了解 windows 之中包含“Input Method Editor (IME)”和“Text Services Framework (TSF)”两套输入法接口。对于使用不同框架的输入法应采用不同方式去获取候选列表。

本章重点讲述 TSF 框架下的输入法候选列表获取。TSF 框架的输入法实际是一个 COM 程序，所以在继续阅读之前，请务必了解 COM 的工作机制。

6.2 关于 IME 候选列表

在网络上应该能搜出一大片此类文章，在此仅做简要说明。 IME 的所有函数都在 imm32.dll 中实现，函数原型可在<imm.h>中查看。

6.2.1 在什么时候获取候选列表信息

如需在程序中自绘候选列表，可在程序中响应如下几个重要消息：

```
switch( uMsg )
{
    case WM_IME_STARTCOMPOSITION:          // 开始编码
    case WM_IME_COMPOSITION:               // 编码串已更新
（显示串更新、上屏串更新、光标位置更改等）
    case WM_IME_ENDCOMPOSITION:            // 输入结束
    case WM_IME_NOTIFY:                    // 这个消息应根据
wParam 的值做如下处理
        switch (wParam)
```

```
            {
            case IMN_OPENCANDIDATE:              // 打开候选列表
            case IMN_CHANGECANDIDATE:            // 更新候选列表
            case IMN_CLOSECANDIDATE:             // 关闭候选列表
            }
        break;
    }
```

2.1.1. 在 WM_IME_COMPOSITION 处获取显示编码串信息， 关键代码如下：

```
case WM_IME_COMPOSITION:
    {
        LONG lRet;
        HIMC hIMC;
        if(hIMC = ImmGetContext(hWnd))
        {
            TCHAR szCompStr[256];
            DWORD dwCursorPos;
            //获取显示字符串
            if ( lParam & GCS_COMPSTR )
            {
                lRet = ImmGetCompositionString( hIMC, GCS_COMPSTR,
szCompStr, ARRAYSIZE( szCompStr ) ) / sizeof(TCHAR);
                szCompStr[lRet] = 0;
            }
            //获取显示字符串的属性标记
            if ( lParam & GCS_COMPSTR )
            {
                lRet = ImmGetCompositionString( hIMC ,
GCS_COMPATTR , szCompStr , ARRAYSIZE( szCompStr ) ) /
sizeof(TCHAR);
                szCompStr[lRet] = 0;
            }
            //获取显示字符串的光标位置
            dwCursorPos = ImmGetCompositionString(hIMC ,
GCS_CURSORPOS, NULL, 0);
            //获取上屏字符串
            if ( lParam & GCS_RESULTSTR )
            {
                lRet = ImmGetCompositionString( hIMC ,
GCS_RESULTSTR , szCompStr , ARRAYSIZE( szCompStr ) ) /
sizeof(TCHAR);
```

```
                szCompStr[lRet] = 0;
            }
            ImmReleaseContext(hWnd,  hIMC);
        }
    }
```

2.1.2. 在 CHANGECANDIDATE 处获取候选列表信息， 关键代码如下：

```
case WM_IME_NOTIFY:
    switch (wParam)
    {
        case IMN_OPENCANDIDATE:              // 打开候选列表
        case IMN_CHANGECANDIDATE:            // 更新候选列表
        {
            HIMC hIMC;
            if (hIMC = ImmGetContext(hWnd))
            {
                LPCANDIDATELIST lpCandList = NULL;
                DWORD dwIndex = 0;
                DWORD  dwBufLen  =  ImmGetCandidateList(hIMC  ,
dwIndex,  NULL,  0 );
                if ( dwBufLen )
                {
                    lpCandList = (LPCANDIDATELIST)GlobalAlloc(GPTR,
dwBufLen);
                    dwBufLen = ImmGetCandidateList(hIMC,  dwIndex,
&lpCandList,  dwBufLen );
                }

                if ( lpCandList )
                {
                    DWORD dwSelection = lpCandList->dwSelection;      //
处于选中状态的候选序号
                    DWORD  dwCount           =  lpCandList->dwCount;
//当前页候选数
                    DWORD  dwPageStart  =  lpCandList->dwPageStart;
//当前页起始序号
                    DWORD  dwPageSize   =  lpCandList->dwPageSize;
//当前页容量
```

```
                    DWORD i;
                    for (i = 0; i < dwCount; i++)
                    {
                        LPTSTR lpCandiString =
(LPTSTR)((DWORD)lpCandList + lpCandList->dwOffset[i]);  //候选字符串
                    }
                    GlobalFree(lpCandList);
                }
                ImmReleaseContext(hWnd，hIMC);
            }
        }
    }
```

6.2.2 TSF 候选列表获取方式

与 TSF 不同，TSF 框架不是基于 IME 消息，而是基于 Sink Event 的机制。TSF 框架允许应用程序在输入框架之上，创建自己的各种 Event 的 Sink。比如转换状态（标点/全角等）改变、输入状态改变（开始输入、刷新输入、结束输入）等。

输入法程序在候选列表发生改变时使用 ITfUIElementMgr::BeginUIElement 、ITfUIElementMgr::UpdateUIElement、ITfUIElementMgr::EndUIElement 等三个接口函数通知 TSF 服务程序，TSF 服务程序再将此类事件转发给所有的 UIElement Sink。

这期间的过程十分复杂，而且大部分步骤都是由输入法程序和操作系统自己完成的。此过程的细节本文不做探讨，仅重点介绍如何创建 Sink、并且如何在 Sink 响应中计算候选列表。

6.2.2.1 注册 Skin 用于接收 BeginUIElement UpdateUIElement EndUIElement 事件

注册 Sink 的关键代码

```
//
//  注册各种 sink
```

```
//
BOOL CandidateReader::SetupSinks()
{
    m_TsfSink = new CUIElementSink();
    hr = m_tm->QueryInterface(__uuidof(ITfSource), (void**)&srcTm);
    hr = srcTm->AdviseSink(IID_ITfThreadMgrEventSink ,
(ITfThreadMgrEventSink*)m_TsfSink, &m_dwThreadMgrCookie);
    hr = srcTm->AdviseSink(__uuidof(ITfUIElementSink) ,
(ITfUIElementSink*)m_TsfSink, &m_dwUIElementSinkCookie);
    hr = srcTm->AdviseSink(__uuidof(ITfInputProcessorProfileActivationSink),
(ITfInputProcessorProfileActivationSink*)m_TsfSink, &m_dwAlpnSinkCookie);
    ITfDocumentMgr* doc_mgr = NULL;
    m_tm->GetFocus(&doc_mgr);
    m_TsfSink->UpdateTextEditSink(doc_mgr);
    doc_mgr->Release();
    srcTm->Release();
}

//
// 重载输入法候选列表更新的 UpdateUIElement 接口
//
STDAPI CUIElementSink::UpdateUIElement(DWORD dwUIElementId)
{
    ITfUIElement* pElement = GetUIElement(dwUIElementId);
    if (!pElement)
        return E_INVALIDARG;

    ITfReadingInformationUIElement* preading = NULL;
    ITfCandidateListUIElement* pcandidate = NULL;
    if (!g_bCandList &&
SUCCEEDED(pElement->QueryInterface(__uuidof(ITfReadingInformationUIE
lement),
        (void**)&preading)))
    {
        MakeReadingInformationString(preading);
        preading->Release();
    }
    else if
(SUCCEEDED(pElement->QueryInterface(__uuidof(ITfCandidateListUIEleme
nt),
        (void**)&pcandidate)))
    {
        MakeCandidateStrings(pcandidate);
        pcandidate->Release();
```

```
    }

    pElement->Release();
    return S_OK;
}
```

6.2.2.2 在 MakeCandidateStrings 中获取候选列表

获取候选列表字符串之前必须首先获取**当前页序号**(uCurrentPage)和**候选列表页索引**(PageIndex)，由于微软一直没有公布 TSF 框架的详细说明文档， 大家都只能从微软的类成员命名中猜测**候选列表页索引**(PageIndex)的数据结构以及使用方式。

所以 pcandidate->GetPageIndex()引起很多程序编写者的误解，在此对其详细说明如下：

- 使用空指针参数调用 pcandidate->GetPageIndex(NULL， 0， &uPageCnt);以获取当前候选列表页数， 候选列表页数会被写入到 uPageCnt 中
- 初始化一个内存块用于保存候选列表每一页的起始序号，内存快大小应为页数*sizof(UINT) IndexList = (UINT *)ImeUiCallback_Malloc(sizeof(UINT)*uPageCnt);
- 再次调用 pcandidate->GetPageIndex(IndexList， uPageCnt， &uPageCnt);
- 调用 pcandidate->GetString(i， &bstr)) 获取候选列表字符串.此处应注意 i 的取值，在上面的代码中如果调用成功，在 IndexLis 会保存后续列表每一页的序号。本文假设候选列表有 16 个候选，每页显示 5 个候选，函数执行完毕时， 各变量会被赋值如下：

```
uPageCnt = 4          //候选列表有 4 页
IndexList[0] = 0      //候选列表第一页的起始序号为 0
IndexList[1] = 5      //候选列表第二页的起始序号为 5
IndexList[2] = 10
IndexList[3] = 15
```

```
void    CandidateReader::MakeCandidateStrings(ITfCandidateListUIElement*
pcandidate)
{
    UINT *pIndexList = NULL;
                    //获取当前选中状态的候选序号（可设置高亮显示，一般为第
一候选）
    pcandidate->GetSelection(&g_Candidate.uIndex);

    //当前候选列表总数
    pcandidate->GetCount(&g_Candidate.uCount);
```

```
    //当前候选列表所在的页
    pcandidate->GetCurrentPage(&g_Candidate.uCurrentPage);

    g_bCandList = true;

    //获取候选列表页每一页对应的起始序号
    pcandidate->GetPageIndex(NULL, 0, &g_Candidate.uPageCnt);
    if (g_Candidate.uPageCnt > 0)
    {
        pIndexList = (UINT *)malloc(sizeof(UINT)*g_Candidate.uPageCnt);
        if (pIndexList)
        {
            pcandidate->GetPageIndex(pIndexList, g_Candidate.uPageCnt, &g_Candidate.uPageCnt);
            g_Candidate.dwPageStart = pIndexList[g_Candidate.uCurrentPage];
            g_Candidate.dwPageSize = (g_Candidate.uCurrentPage < g_Candidate.uPageCnt - 1) ?
                min(g_Candidate.uCount, pIndexList[g_Candidate.uCurrentPage + 1]) - g_Candidate.dwPageStart :
                g_Candidate.uCount - g_Candidate.dwPageStart;
        }
    }

    // 本示例的 g_Candidate.szCandidate 最大个数为 10, 因此 min 处理
    UINT uCandPageSize = min(g_Candidate.dwPageSize, 10);
    for (UINT i = g_Candidate.dwPageStart, j = 0; (DWORD)i < g_Candidate.uCount && j < 128; i++, j++)
    {
        //获取候选列表的第 i 个候选串
        if (SUCCEEDED(pcandidate->GetString(i, &bstr)))
        {
            if (bstr)
            {
                wcscpy_s(g_Candidate.szCandidate[j], bstr);

                // 应注意 TSF 框架要求输入内部必须使用 SysAlloc() 分配候选列表字符串保存空间
                // 在调用 pcandidate->GetString 之后, 必须使用 SysFreeString(bstr)释放
                SysFreeString(bstr);
            }
        }
    }
```

```
    }
```

6.2.3 本章的完整代码

https://github.com/chinput/InputMethodCandidateReader