ActiveX控件安全警告

|  |
| --- |
| 在tryISafeObjectCtrl.h中定义ISafeObject接口：  #include <objsafe.h> // for IObjectSafety; in ActiveX SDK  class CtryISafeObjectCtrl : public COleControl {   DECLARE\_DYNCREATE(CtryISafeObjectCtrl) //........................................................................ //ISafeObject  DECLARE\_INTERFACE\_MAP()   BEGIN\_INTERFACE\_PART(ObjSafe, IObjectSafety)   STDMETHOD\_(HRESULT, GetInterfaceSafetyOptions) (              /\* [in] \*/ REFIID riid,             /\* [out] \*/ DWORD \_\_RPC\_FAR \*pdwSupportedOptions,             /\* [out] \*/ DWORD \_\_RPC\_FAR \*pdwEnabledOptions   );                  STDMETHOD\_(HRESULT, SetInterfaceSafetyOptions) (              /\* [in] \*/ REFIID riid,             /\* [in] \*/ DWORD dwOptionSetMask,             /\* [in] \*/ DWORD dwEnabledOptions   );  END\_INTERFACE\_PART(ObjSafe);    //ISafeObject //........................................................................  。。。。。  };  在objsafe.h头文件中有ISafeObject接口的相关定义  3。在tryISafeObjectCtrl.cpp中ISafeObject接口的相关实现：  //............................................................................. // Interface map for IObjectSafety  BEGIN\_INTERFACE\_MAP( CtryISafeObjectCtrl, COleControl )  INTERFACE\_PART(CtryISafeObjectCtrl, IID\_IObjectSafety, ObjSafe) END\_INTERFACE\_MAP()  //............................................................................. // IObjectSafety member functions  // Delegate AddRef, Release, QueryInterface  ULONG FAR EXPORT CtryISafeObjectCtrl::XObjSafe::AddRef() {     METHOD\_PROLOGUE(CtryISafeObjectCtrl, ObjSafe)     return pThis->ExternalAddRef(); }  ULONG FAR EXPORT CtryISafeObjectCtrl::XObjSafe::Release() {     METHOD\_PROLOGUE(CtryISafeObjectCtrl, ObjSafe)     return pThis->ExternalRelease(); }  HRESULT FAR EXPORT CtryISafeObjectCtrl::XObjSafe::QueryInterface(     REFIID iid, void FAR\* FAR\* ppvObj) {     METHOD\_PROLOGUE(CtryISafeObjectCtrl, ObjSafe)     return (HRESULT)pThis->ExternalQueryInterface(&iid, ppvObj); }  const DWORD dwSupportedBits =    INTERFACESAFE\_FOR\_UNTRUSTED\_CALLER |   INTERFACESAFE\_FOR\_UNTRUSTED\_DATA; const DWORD dwNotSupportedBits = ~ dwSupportedBits;  //............................................................................. // CStopLiteCtrl::XObjSafe::GetInterfaceSafetyOptions // Allows container to query what interfaces are safe for what. We're // optimizing significantly by ignoring which interface the caller is // asking for. HRESULT STDMETHODCALLTYPE   CtryISafeObjectCtrl::XObjSafe::GetInterfaceSafetyOptions(    /\* [in] \*/ REFIID riid,         /\* [out] \*/ DWORD \_\_RPC\_FAR \*pdwSupportedOptions,         /\* [out] \*/ DWORD \_\_RPC\_FAR \*pdwEnabledOptions) {  METHOD\_PROLOGUE(CtryISafeObjectCtrl, ObjSafe)   HRESULT retval = ResultFromScode(S\_OK);   // does interface exist?  IUnknown FAR\* punkInterface;  retval = pThis->ExternalQueryInterface(&riid,       (void \* \*)&punkInterface);  if (retval != E\_NOINTERFACE) { // interface exists   punkInterface->Release(); // release it--just checking!  }    // we support both kinds of safety and have always both set,  // regardless of interface  \*pdwSupportedOptions = \*pdwEnabledOptions = dwSupportedBits;   return retval; // E\_NOINTERFACE if QI failed }  ///////////////////////////////////////////////////////////////////////////// // CStopLiteCtrl::XObjSafe::SetInterfaceSafetyOptions // Since we're always safe, this is a no-brainer--but we do check to make // sure the interface requested exists and that the options we're asked to // set exist and are set on (we don't support unsafe mode). HRESULT STDMETHODCALLTYPE   CtryISafeObjectCtrl::XObjSafe::SetInterfaceSafetyOptions(          /\* [in] \*/ REFIID riid,         /\* [in] \*/ DWORD dwOptionSetMask,         /\* [in] \*/ DWORD dwEnabledOptions) {     METHOD\_PROLOGUE(CtryISafeObjectCtrl, ObjSafe)    // does interface exist?  IUnknown FAR\* punkInterface;  pThis->ExternalQueryInterface(&riid, (void \* \*)&punkInterface);  if (punkInterface) { // interface exists   punkInterface->Release(); // release it--just checking!  }  else { // interface doesn't exist   return ResultFromScode(E\_NOINTERFACE);  }   // can't set bits we don't support  if (dwOptionSetMask & dwNotSupportedBits) {    return ResultFromScode(E\_FAIL);  }    // can't set bits we do support to zero  dwEnabledOptions &= dwSupportedBits;  // (we already know there are no extra bits in mask )  if ((dwOptionSetMask & dwEnabledOptions) !=    dwOptionSetMask) {   return ResultFromScode(E\_FAIL);  }            // don't need to change anything since we're always safe  return ResultFromScode(S\_OK); } |
| 定义GUID：  const GUID CDECL BASED\_CODE CLSID\_SafeItem =    { 0x94fd3218, 0x4b05, 0x4385, {0x81, 0x2b, 0x55, 0xc0, 0xf4, 0x6e, 0xd6, 0x6} }; /\*它即是：ECActiveCtrl.cpp文件中的，  // 初始化类工厂和 guid  IMPLEMENT\_OLECREATE\_EX(CECActiveCtrl, "ECACTIVE.ECActiveCtrl.1", 0x94fd3218, 0x4b05, 0x4385, 0x81, 0x2b, 0x55, 0xc0, 0xf4, 0x6e, 0xd6, 0x6) 转化得来的！\*/  添加方法：  // 创建组件种类 HRESULT CreateComponentCategory(CATID catid, WCHAR\* catDescription) {     ICatRegister\* pcr = NULL ;     HRESULT hr = S\_OK ;     hr = CoCreateInstance(CLSID\_StdComponentCategoriesMgr, NULL, CLSCTX\_INPROC\_SERVER, IID\_ICatRegister, (void\*\*)&pcr); if (FAILED(hr))            return hr;     // Make sure the HKCR\Component Categories\{..catid}     // key is registered.     CATEGORYINFO catinfo;     catinfo.catid = catid;     catinfo.lcid = 0x0409 ; // english     // Make sure the provided description is not too long.     // Only copy the first 127 characters if it is.     int len = (int)wcslen(catDescription);     if (len>127)            len = 127;     wcsncpy\_s(catinfo.szDescription, 128, catDescription, len);     // Make sure the description is null terminated.     catinfo.szDescription[len] = '\0';     hr = pcr->RegisterCategories(1, &catinfo);     pcr->Release();     return hr; }  // 注册组件种类 HRESULT RegisterCLSIDInCategory(REFCLSID clsid, CATID catid) {     // Register your component categories information.     ICatRegister\* pcr = NULL ;     HRESULT hr = S\_OK ;     hr = CoCreateInstance(CLSID\_StdComponentCategoriesMgr, NULL, CLSCTX\_INPROC\_SERVER, IID\_ICatRegister, (void\*\*)&pcr);     if (SUCCEEDED(hr))     {            // Register this category as being implemented by the class.           CATID rgcatid[1] ;            rgcatid[0] = catid;            hr = pcr->RegisterClassImplCategories(clsid, 1, rgcatid);     }     if (pcr != NULL)            pcr->Release();     return hr; }  // 卸载组件种类 HRESULT UnRegisterCLSIDInCategory(REFCLSID clsid, CATID catid) {     ICatRegister\* pcr = NULL ;     HRESULT hr = S\_OK ;    hr = CoCreateInstance(CLSID\_StdComponentCategoriesMgr, NULL, CLSCTX\_INPROC\_SERVER, IID\_ICatRegister, (void\*\*)&pcr);     if (SUCCEEDED(hr))     {            // Unregister this category as being implemented by the class.            CATID rgcatid[1] ;            rgcatid[0] = catid;            hr = pcr->UnRegisterClassImplCategories(clsid, 1, rgcatid);     }     if (pcr != NULL)            pcr->Release();    return hr; }  修改方法DllRegisterServer如下：  STDAPI DllRegisterServer(void) { HRESULT hr; AFX\_MANAGE\_STATE(\_afxModuleAddrThis);  if (!AfxOleRegisterTypeLib(AfxGetInstanceHandle(), \_tlid))    return ResultFromScode(SELFREG\_E\_TYPELIB);  if (!COleObjectFactoryEx::UpdateRegistryAll(TRUE))    return ResultFromScode(SELFREG\_E\_CLASS);  // 标记控件初始化安全.     // 创建初始化安全组件种类     hr = CreateComponentCategory(CATID\_SafeForInitializing, L"Controls safely initializable from persistent data!");     if (FAILED(hr))            return hr;     // 注册初始化安全     hr = RegisterCLSIDInCategory(CLSID\_SafeItem, CATID\_SafeForInitializing);     if (FAILED(hr))            return hr;     // 标记控件脚本安全     // 创建脚本安全组件种类      hr = CreateComponentCategory(CATID\_SafeForScripting, L"Controls safely scriptable!");     if (FAILED(hr))            return hr;     // 注册脚本安全组件种类     hr = RegisterCLSIDInCategory(CLSID\_SafeItem, CATID\_SafeForScripting);     if (FAILED(hr))            return hr;  return NOERROR; }  修改方法DllUnregisterServer如下：  STDAPI DllUnregisterServer(void) { HRESULT hr; AFX\_MANAGE\_STATE(\_afxModuleAddrThis);  if (!AfxOleUnregisterTypeLib(\_tlid, \_wVerMajor, \_wVerMinor))    return ResultFromScode(SELFREG\_E\_TYPELIB);  if (!COleObjectFactoryEx::UpdateRegistryAll(FALSE))    return ResultFromScode(SELFREG\_E\_CLASS);  // 删除控件初始化安全入口.     hr=UnRegisterCLSIDInCategory(CLSID\_SafeItem, CATID\_SafeForInitializing);     if (FAILED(hr))            return hr;     // 删除控件脚本安全入口     hr=UnRegisterCLSIDInCategory(CLSID\_SafeItem, CATID\_SafeForScripting);     if (FAILED(hr))            return hr;  return NOERROR; } |