组件通讯分类

方式	备注
点对点	组件=>组件（暂不支持）
组播	组件=>某组类型的组件（支持配置文件路由、支持只发送给当前页面组件）
广播	组件=>所有组件（支持只发送给当前页面组件）

配置设计

组件分组配置

元数据文件（CosmosAppMetadata.xml）的组件信息中增加Group节点，用于给当前组件进行分组（支持属于多个组，用逗号分隔）

```
<?xml version="1.0" encoding="utf-8" ?>
<Cosmos>
  <!-- app元数据 -->
  <App>
    <Version>1.0.0.0</Version>
    <Base>
      <!-- 包id，填入在管理端申请应用是返回的Appid -->
      <Guid>4bdb5aa3-9081-4918-be2b-09102e6a2615</Guid>
      <Name>
        <zh-CN>Cosmos进程demo</zh-CN>
        <en-US>Cosmos进程demo</en-US>
      </Name>
      <Description>
        <zh-CN>Cosmos进程demo</zh-CN>
        <en-US>Cosmos进程demo</en-US>
      </Description>
    </Base>
    <!-- widget元数据集 -->
    <Widgets>
      <Widget>
        <Base>
          <!-- 组件id，填入在管理端申请应用是添加组件返回的Widgetid -->
          <Guid>cdf370d9-8a19-4175-9e1e-31511eaa5017</Guid>
          <Name>
            <zh-CN>Cosmos进程demo</zh-CN>
            <en-US>Cosmos进程demo</en-US>
          </Name>
          <Description>
            <zh-CN>Cosmos进程demo</zh-CN>
            <en-US>Cosmos进程demo</en-US>
          </Description>
        </Base>
        <font color="red"><Group>trader,hq</Group>
        <!-- 组件库路径 -->
        <LibraryFile>Cosmos.App.Hithink.MfcProcessDemo.dll</LibraryFile>
```

```
        <!-- 组件库入口类 -->

<RuntimeType>Cosmos.App.Hithink.MfcProcessDemo.MfcProcessDemoGui</RuntimeType>
        <GuiFrameworks>wpf</GuiFrameworks>
      </Widget>
    </Widgets>
  </App>
</Cosmos>
```

组件布局配置

基础布局文件可通过在方案管理器中拖拽生成布局文件，新增RouterGroup节点，用于配置路由信息

布局文件生成路径：cosmos主程序路径下的WndManager文件夹中，文件名为uuid+tab页名称

配置文件内容如下

```
<?xml version="1.0" encoding="utf-8"?>
<LayoutRoot>
  <RootPanel Orientation="Horizontal">
    <LayoutAnchorablePane DockWidth="1.2921547050595492*" FloatingWidth="400"
FloatingHeight="300" FloatingLeft="148" FloatingTop="528">
      <LayoutAnchorable AutoHideMinWidth="100" AutoHideMinHeight="100"
Title="Cosmos组件demoe" IsSelected="True" ContentId="b0fd068e-2021-4619-acc0-
53cda8d94a37" Id="b0fd068e-2021-4619-acc0-53cda8d94a37_ad4381b4-4b55-4e91-b387-
abe0dcc3a7dc" FloatingLeft="148" FloatingTop="528" FloatingWidth="400"
FloatingHeight="300" CanClose="False" LastActivationTimeStamp="05/14/2025
15:35:30" />
    </LayoutAnchorablePane>
    <LayoutAnchorablePaneGroup Orientation="Vertical"
DockWidth="0.4823181578846188*">
      <LayoutAnchorablePane FloatingWidth="400" FloatingHeight="300"
FloatingLeft="1598" FloatingTop="378">
        <LayoutAnchorable AutoHideMinWidth="100" AutoHideMinHeight="100"
Title="Cosmos进程demo" IsSelected="True" ContentId="cdf370d9-8a19-4175-9e1e-
31511eaa5017" Id="cdf370d9-8a19-4175-9e1e-31511eaa5017_92609b4d-924f-4415-805b-
9448c4dc0c03" FloatingLeft="1598" FloatingTop="378" FloatingWidth="400"
FloatingHeight="300" CanClose="False" LastActivationTimeStamp="05/14/2025
15:35:13" />
      </LayoutAnchorablePane>
      <LayoutAnchorablePane FloatingWidth="400" FloatingHeight="300"
FloatingLeft="1631" FloatingTop="401">
        <LayoutAnchorable AutoHideMinWidth="100" AutoHideMinHeight="100"
Title="WPF进程demo" IsSelected="True" ContentId="8b1cd85f-9143-4b9a-8d5d-
6c1a544a3f3c" Id="8b1cd85f-9143-4b9a-8d5d-6c1a544a3f3c_f0900a67-9a5b-44a8-8d10-
256064a7acbb" FloatingLeft="1631" FloatingTop="401" FloatingWidth="400"
FloatingHeight="300" CanClose="False" LastActivationTimeStamp="05/14/2025
15:35:28" />
      </LayoutAnchorablePane>
      <LayoutAnchorablePane DockHeight="0.6666666666666666*" FloatingWidth="400"
FloatingHeight="300" FloatingLeft="1562" FloatingTop="391">
```

```
        <LayoutAnchorable AutoHideMinWidth="100" AutoHideMinHeight="100"
Title="Cosmos进程demo" IsSelected="True" ContentId="cdf370d9-8a19-4175-9e1e-
31511eaa5017" Id="cdf370d9-8a19-4175-9e1e-31511eaa5017_7719c4f1-ad44-408e-9597-
704fe941a22d" FloatingLeft="1562" FloatingTop="391" FloatingWidth="400"
FloatingHeight="300" CanClose="False" LastActivationTimeStamp="05/14/2025
15:35:06" />
      </LayoutAnchorablePane>
      <LayoutAnchorablePane DockHeight="0.3333333333333333*" FloatingWidth="400"
FloatingHeight="385" FloatingLeft="1605" FloatingTop="748">
        <LayoutAnchorable AutoHideMinWidth="100" AutoHideMinHeight="100"
Title="Cosmos进程demo" IsSelected="True" ContentId="cdf370d9-8a19-4175-9e1e-
31511eaa5017" Id="cdf370d9-8a19-4175-9e1e-31511eaa5017_70373a02-44aa-41ac-9656-
1c51a17c3f10" FloatingLeft="1605" FloatingTop="748" FloatingWidth="400"
FloatingHeight="385" CanClose="False" LastActivationTimeStamp="05/14/2025
15:34:58" />
      </LayoutAnchorablePane>
    </LayoutAnchorablePaneGroup>
  </RootPanel>
  <RouterGroup>
    <Router Id="b0fd068e-2021-4619-acc0-53cda8d94a37_ad4381b4-4b55-4e91-b387-
abe0dcc3a7dc">
      <Trace From="hq" To="cdf370d9-8a19-4175-9e1e-31511eaa5017_92609b4d-924f-
4415-805b-9448c4dc0c03" />
      <Trace From="trader" To="8b1cd85f-9143-4b9a-8d5d-6c1a544a3f3c_f0900a67-
9a5b-44a8-8d10-256064a7acbb" />
    </Router>
    <Router Id="cdf370d9-8a19-4175-9e1e-31511eaa5017_92609b4d-924f-4415-805b-
9448c4dc0c03">
      <Trace From="trader" To="8b1cd85f-9143-4b9a-8d5d-6c1a544a3f3c_f0900a67-
9a5b-44a8-8d10-256064a7acbb" />
    </Router>
  </RouterGroup>
  <TopSide />
  <RightSide />
  <LeftSide />
  <BottomSide />
  <FloatingWindows />
  <Hidden />
</LayoutRoot>
```

配置文件解释：

RootPanel：该节点有方案管理器自动生成，不需要手动配置

RouterGroup：保留当前页面组件的路由信息

Router：每一个Router节点代表一个组件实例的路由信息表， Id属性表示组件实例id，从LayoutAnchorable节点中的Id字段

Trace：该组件实例下的一条路由信息，From属性为来源，To属性表示真正需要路由的实例id

注意：

1. 该配置文件的节点顺序不要调整，会导致配置文件解析失败（举例：RouterGroup节点写到RootPanel之前）
2. RouterGroup中属性的大小写不要写错，会导致解析失败（举例：Router id="b0fd068e-2021-4619-acc0-53cda8d94a37_ad4381b4-4b55-4e91-b387-abe0dcc3a7dc"中 大写的I变成小写的i ）

接口设计

```
/// <summary>
/// 定义Cosmos RPC请求的接口，包含请求ID、方法及参数。
/// 该接口用于标准化RPC请求的基本结构，确保请求对象包含必要元数据。
/// </summary>
public interface ICosmosRpcRequest
{
    /// <summary>
    /// 在进程间通信时代表请求的唯一标识。该ID用于追踪和关联请求与响应。
    /// 在组件间通信时填入发送方的实例id
    /// </summary>
    string id { get; set; }

    /// <summary>
    /// 获取或设置要调用的RPC方法名称。
    /// 方法名应与服务端支持的接口严格匹配。
    /// </summary>
    string method { get; set; }

    /// <summary>
    /// 获取或设置请求参数。
    /// 使用JToken类型可支持动态JSON结构，方便处理不同复杂度的参数。
    /// </summary>
    JToken param { get; set; }
}

/// <summary>
/// 应答参数
/// </summary>
public interface ICosmosRpcResponse
{
    /// <summary>
    /// 在进程间通信时代表请求的唯一标识。该ID用于追踪和关联请求与响应。
    /// 在组件间通信时填入应答方的实例id
    /// </summary>
    string id { get; set; }

    /// <summary>
    /// 错误吗
    /// 进程间通信时
    /// code = 0 表示业务返回成功，code != 0 表示业务返回错误，业务按需定义错误编码
    /// 组件间通信时
    /// code = 0 表示业务返回成功
    /// code = 1 表示填入的发送方实例找不到
    /// code = 2 表示填入的接收方实例找不到
    /// code = 3 全局或者给某组组件发消息时，没有找到符合条件的组件时
    /// </summary>
    int code { get; set; }

    /// <summary>
    /// 应答错误时，返回错误信息。
    /// </summary>
    JToken error { get; set; }

    /// <summary>
```

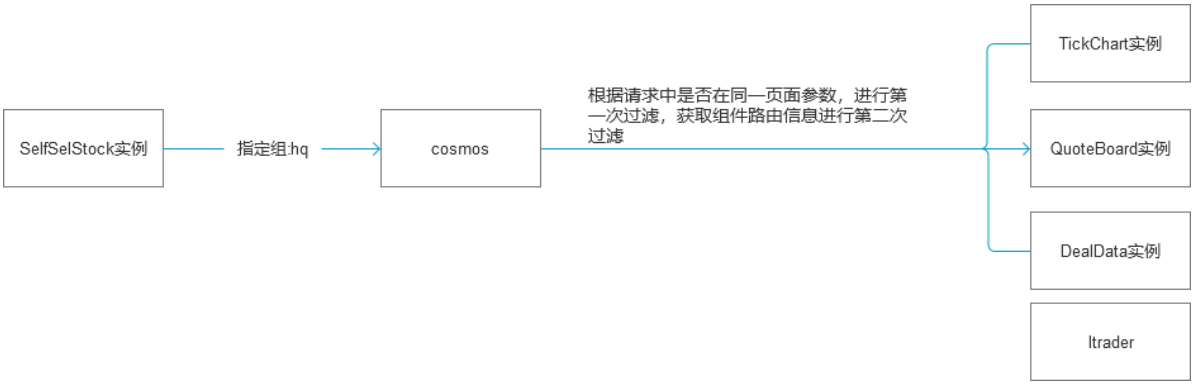
```
    /// 应答成功时，返回结果信息。
    /// </summary>
    JToken result { get; set; }
}

/// <summary>
/// 组件引擎提供的业务请求类接口。
/// 此接口定义了与业务相关的请求操作。
/// </summary>
public interface ICosmosBusinessRequest
{
    /// <summary>
    /// 通信接口-发送消息到指定组件
    /// </summary>
    /// <param name="Instanceid">发送对象，根据type填入不同的参数</param>
    /// <param name="type">发送对象类型</param>
    /// <param name="parameter">发送内容</param>
    /// <param name="currentPage">是否只发送给当前页面</param>
    Task<ICosmosRpcResponse> InvokeWidget(string Instanceid, InvokeType type,
ICosmosRpcRequest parameter, bool currentPage = false);

    /// <summary>
    /// 通信接口-通知消息到指定组件
    /// </summary>
    /// <param name="Instanceid">发送对象，根据type填入不同的参数</param>
    /// <param name="type">发送对象类型</param>
    /// <param name="parameter">发送内容</param>
    /// <param name="currentPage">是否只发送给当前页面</param>
    Task NotifyWidget(string Instanceid, InvokeType type, ICosmosRpcRequest
parameter, bool currentPage = false);
}

/// <summary>
/// 处理进程通信调用类
/// </summary>
public interface ICosmosWidgetComunication
{
    /// <summary>
    /// 处理外部组件调用请求
    /// </summary>
    /// <param name="parameter"></param>
    /// <returns></returns>
    Task<ICosmosRpcResponse> OnInvoke(ICosmosRpcRequest parameter);

    /// <summary>
    /// 处理外部组件通知
    /// </summary>
    /// <param name="parameter"></param>
    /// <returns></returns>
    void OnNotify(ICosmosRpcRequest parameter);
}
```

1.

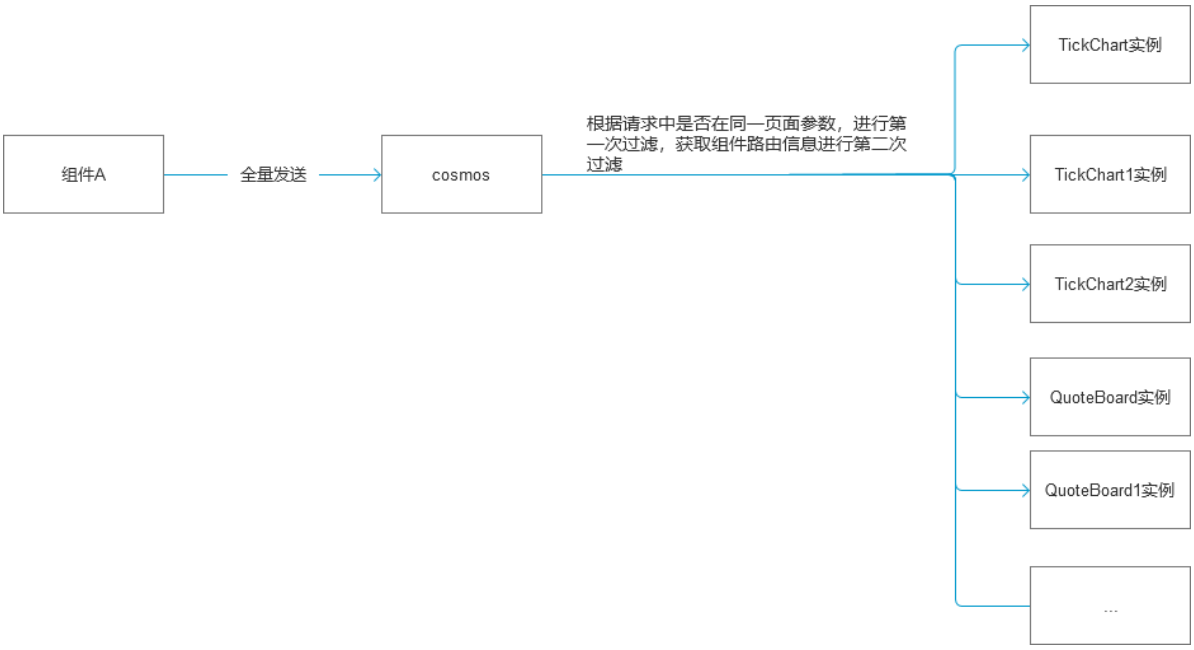
交互图

进程内

给某组组件发送消息



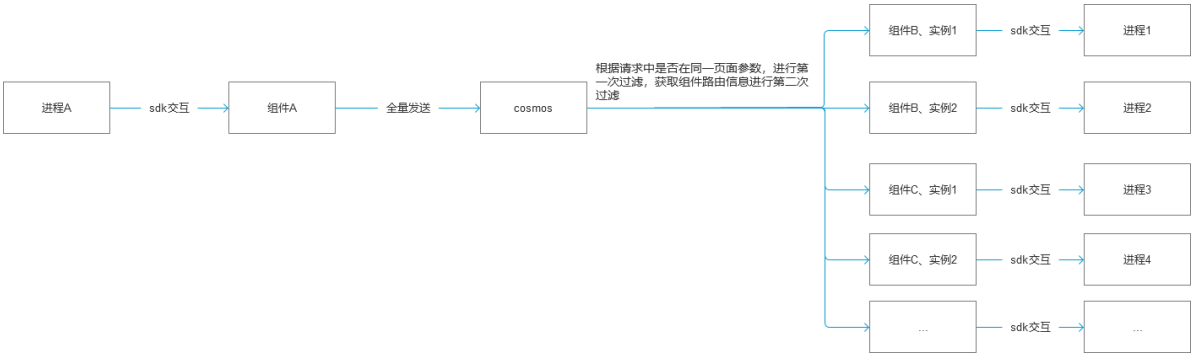
给所有组件发送消息



进程间

给某组组件发送消息

给所有组件发送消息



示例 参考示例

向某类组件发送消息

```
var request =
_contextInjection.ThisAppContext.GlobalContexts.EngineContext.BusinessRequest.Cr
eateRequestParameter();
request.method = "textchanged";
request.id = _contextInjection.ThisInstanceContext.Id;
request.param = new JObject()
{
    ["text"] = text_sendcom.Text
};

//发送请求
if (comuType.SelectedIndex == 0)
{
    var result =
_contextInjection.ThisAppContext.GlobalContexts.EngineContext.BusinessRequest.In
vokeWidget(text_sender.Text, InvokeType.Group, request, true);
}
//发送通知
else
{

 _contextInjection.ThisAppContext.GlobalContexts.EngineContext.BusinessRequest.N
otifyWidget(text_sender.Text, InvokeType.Group, request, true);
}
```

给所有组件发送消息

```
var request =
_contextInjection.ThisAppContext.GlobalContexts.EngineContext.BusinessRequest.Cr
eateRequestParameter();
request.method = "textchanged";
request.id = _contextInjection.ThisInstanceContext.Id;
request.param = new JObject()
{
    ["text"] = text_sendcom.Text
};
//发送请求
if( comuType.SelectedIndex == 0 )
{
```

```
    var result =
_contextInjection.ThisAppContext.GlobalContexts.EngineContext.BusinessRequest.In
vokeWidget(null, InvokeType.Global, request, true);
}
//发送通知
else
{

 _contextInjection.ThisAppContext.GlobalContexts.EngineContext.BusinessRequest.N
otifyWidget(null, InvokeType.Global, request, true);
}
```

接受其他组件发送的消息（继承ICosmosWidgetComunication类）

处理调用

```
  public Task<ICosmosRpcResponse> OnInvoke(ICosmosRpcRequest parameter)
  {
      ICosmosRpcResponse response =
ContextInjection.ThisAppContext.GlobalContexts.EngineContext.BusinessRequest.Cre
ateResponseParameter();
      Console.WriteLine($"WpfComDemoGui 接收到其他组件发起请求  请求方法为:
{parameter.method} ，参数为 {parameter.param.ToString()}, 来源{parameter.id}");
      response.code = 200;
      return Task.FromResult(response);
  }
```

处理推送

```
  public void OnNotify(ICosmosRpcRequest parameter)
  {
      Console.WriteLine($"WpfComDemoGui 接收到其他组件发起通知 通知方法为 :
{parameter.method} ，参数为 {parameter.param.ToString()}, 来源{parameter.id}");
  }
```

进程间组件接入

一：继承WpfCosmosAppProcessWidget类 参考示例

```
namespace Cosmos.App.Hithink.MfcProcessDemo
{
    public class MfcProcessDemoGui :
        WpfCosmosAppProcessWidget //进程间通讯基类，并且需要实现类中提供的抽象方法。

    {
    }
 }
```

二：实现接口

```
/// <summary>
/// 接受其他组件调用方法
/// </summary>
public override Task<ICosmosRpcResponse> onInvoke(ICosmosRpcRequest parameter,
ref bool bHandle)
{
    //组件是否需要处理进程间通讯请求，如果要处理bHandle置为true，并且执行自己的业务代码，处理
完成后返回resoinse、如果不处理返回null即可

    /*//处理进程间通讯请求,不往进程发送
    {
        bHandle = true;
        ICosmosRpcResponse response =
_ContextInjection.ThisAppContext.GlobalContexts.EngineContext.BusinessRequest.Cr
eateResponseParameter();
        Console.WriteLine($"MfcProcessDemoGui 接收到其他组件发起请求  请求方法为:
{parameter.method} ，参数为 {parameter.param.ToString()}, 来源{parameter.id}");
        response.code = 200;
        return Task.FromResult(response);
    }

    //处理进程间通讯请求,且进程发送
    {
        ICosmosRpcResponse response =
_ContextInjection.ThisAppContext.GlobalContexts.EngineContext.BusinessRequest.Cr
eateResponseParameter();
        Console.WriteLine($"MfcProcessDemoGui 接收到其他组件发起请求  请求方法为:
{parameter.method} ，参数为 {parameter.param.ToString()}, 来源{parameter.id}");
        response.code = 200;
        return Task.FromResult(response);
    }*/

    //不处理进程通讯进球，让进程自己处理
    {
        return null;
    }
}

/// <summary>
/// 接受其他组件通知
/// </summary>
public override void onNotify(ICosmosRpcRequest parameter, ref bool bHandle)
{
    //组件是否需要单独处理进程间通讯通知，如果要处理bHandle置为true，并且执行自己的业务代码、
如果不处理返回null即可

    /* //处理进程间通讯请求,不往进程发送
     {
         bHandle = true;
         Console.WriteLine($"MfcProcessDemoGui 接收到其他组件发起通知 通知方法为:
{parameter.method} ，参数为 {parameter.param.ToString()}, 来源{parameter.id}");
     }
```

```
     //处理进程间通讯请求,且进程发送
     {
         Console.WriteLine($"MfcProcessDemoGui 接收到其他组件发起通知 通知方法为:
{parameter.method} ，参数为 {parameter.param.ToString()}, 来源{parameter.id}");
     }*/

    //不处理进程通讯进球，让进程自己处理
    {
        return;
    }
}

/// <summary>
/// 处理推送数据（进程间消息）
/// </summary>
protected override void HandlePush(ICosmosRpcPush data)
{
    string topic = data.topic;
    if (topic == "push_account")
    {
        // 处理推送账户信息
        string strID = (string)data.param["ID"];
        if (g_mapActInfo.ContainsKey(strID))
        {
            g_mapActInfo[strID].Type = (int)data.param["Type"];
            g_mapActInfo[strID].Status = (int)data.param["Status"];
        }
    }
}

/// <summary>
/// 处理通知（进程间消息）
/// </summary>
protected override void HandleNotify(ICosmosRpcRequest data)
{
    string method = data.method;
    if (method == "init_succ")
    {
        // 对方进程初始化完成，按需实现业务需求
        ChangeProcessWindowSize();
    }
    else if (method == "notf_sub")
    {
        // 对方初始化成功，可以向其发送通信，这里做如下业务操作
        // 1、订阅账户信息
        ICosmosRpcRequest sub = CreateRpcRequest();
        sub.id = Guid.NewGuid().ToString();
        sub.method = "sub_account";
        sub.param = new JObject
        {
            ["ID"] = "123456" //订阅账户123456信息
        };

        g_mapRequest[sub.id] = sub;
        RpcSubscribeAsync(sub);


        // 2、查询账户信息（同步调用）
```

```
        ICosmosRpcRequest sync_param = CreateRpcRequest();
        sync_param.id = Guid.NewGuid().ToString();
        sync_param.method = "qry_account";
        sync_param.param = new JObject
        {
            ["ID"] = "ALL" //查询所有账户信息
        };

        g_mapRequest[sync_param.id] = sync_param;
        ICosmosRpcResponse sync_ret = RpcInvokeAsync(sync_param).Result; //默认30
秒超时
        {
            // 处理返回结果 sync_ret
            if (g_mapRequest.ContainsKey(sync_ret.id))
            {
                // 找到了请求上下文
                if (sync_ret.code == 0)
                {
                    // 返回业务成功
                    string async_method = g_mapRequest[sync_ret.id].method;
                    if (async_method == "qry_account")
                    {
                        JToken resultToken = sync_ret.result;

                        // 反序列化 result 属性
                        var deserializedAccounts =
JsonConvert.DeserializeObject<List<AccountInfo>>(resultToken.ToString());
                        foreach (var account in deserializedAccounts)
                        {
                            g_mapActInfo[account.ID] = account;
                        }
                    }
                }
                else
                {
                    // 返回业务报错
                    // 按报错处理
                }
            }
        }

        // 3、查询账户信息（异步调用）
        QryAccount_InvokeAsync();
    }
}

/// <summary>
/// 处理调用（进程间消息）
/// </summary>
protected override async Task<ICosmosRpcResponse> HandleInvoke(ICosmosRpcRequest
data)
{
    ICosmosRpcResponse resp = CreateRpcResponse();
    if (data.method == "requestTheme")
    {
        resp.code = 0;
        var theme =
_ContextInjection.ThisAppContext.GlobalContexts.VisualContext.ColorScheme;
```

```
        resp.result = new JObject();
        resp.result["theme"] = theme;
    }
    else if (data.method.Contains("requestThemeRes"))
    {
        resp.code = 0;
        var dictionary =
_ContextInjection.ThisAppContext.GlobalContexts.VisualContext.ThemeResources;
        resp.result = JToken.FromObject(dictionary);
    }
    else if (data.method == "test_invoke")
    {
        System.Threading.Thread.Sleep(5000);
        resp.code = 0;
        resp.result = new JObject();
        resp.result["theme"] = "onresp_test_invoke";
    }
    else if (data.method == "test_invoke_async")
    {
        System.Threading.Thread.Sleep(5000);
        resp.code = 0;
        resp.result = new JObject();
        resp.result["theme"] = "onresp_test_invoke_async";
    }

    return resp;
}

/// <summary>
/// 处理订阅（进程间消息）
/// </summary>
protected override void HandleSubscribe(ICosmosRpcRequest data)
{
    // 处理 客户端发送的 SUBS 事件（注：业务按需处理，以下为示例）
}
```