23 异步服务

Phenix丶的 RPC 服务，默认下是同步（阻塞式）的，也就是客户端的调用进程，需保持睡眠状态直到服务端返回应答信息为止。如果服务端的运算需时很长（比如 1 分钟以上），为了客户的良好体验感受，还是建议使用异步服务的模式。这样，在服务运算期间，客户端可以获知服务端的运算进度，也可以干预到服务端的进程，比如通知服务端中断运算中的进程。

23.1 实现方法

23.1.1 Phenix.Core.Data.ServiceBase<T>类

ServiceBase<T>服务类，即提供同步服务模式，也提供了异步服务模式。

具体方法如下：

方法	说明	参数类型	参数	参数说明
Execute	同步执行指令			
ExecuteAsync	异步执行指令	Action<StopEventArgs>	doProgressAsking	进度问讯前事件
		Action<ShallEventArgs>	doProgressAsked	进度问讯到事件
		int	askProgressInterval	询问进度间隔(秒)
StopAsync	中止异步指令			

相关的属性有：

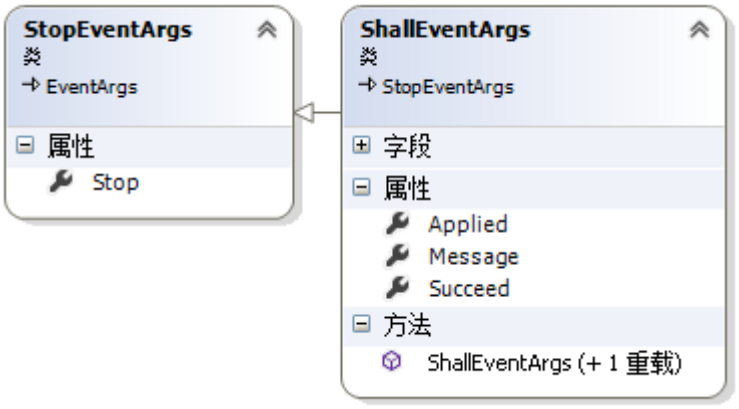
属性	说明	备注
ExecuteResult	执行结果	类型：ShallEventArgs
AskProgressInterval	询问进度间隔(秒)	默认：3 秒
InAsync	是否处于异步状态	

23.1.2 异步回调函数及其传参类型

异步执行指令函数所传入的回调函数，用于处理在客户端每隔多少（askProgressInterval）秒来异步询问服务端之前（doProgressAsking）的事件和之后（doProgressAsked）的事件。

回调函数 doProgressAsking 提供了向服务端发出中止服务（StopEventArgs.Stop）信息的机会，而回调函数 doProgressAsked 则提供了处理从服务端反馈信息（ShallEventArgs.Message）的机会。

以上涉及到的 StopEventArgs 类和 ShallEventArgs 类，被定义如下：



属性	说明	备注
Stop	是否终止	提供中止服务的机会；
Applied	是否已应用	本服务功能中无意义；
Succeed	是否已成功	内部使用，请不要在业务代码中赋值；在服务端代码执行时一直为 false，只有当服务端的执行过程（DoExecute）结束后，或者被客户端要求中止时，才会变为 true；
Message	消息对象	服务端的执行过程（DoExecute）可以随时把中间结果、最终结果赋值给本对象，由 Phenix 传回给客户端；

23.1.3 服务端的业务代码编写位置

以上服务端的执行过程（DoExecute 函数），就是我们编写服务端的业务代码的地方，需要被重载覆写：

```
/// <summary>
/// 处理执行指令
/// 在运行持久层的程序域里被调用
/// 请使用Phenix.Core.Data.DbCommandHelper与数据库交互以保证可扩展性
/// 如果重载了DoExecute()且未调用base.DoExecute()，则本函数将不会执行到
/// </summary>
/// <param name="connection">数据库连接</param>
protected virtual void DoExecute(DbConnection connection) { }

/// <summary>
/// 处理执行指令
/// 在运行持久层的程序域里被调用
/// 请使用Phenix.Core.Data.DbCommandHelper与数据库交互以保证可扩展性
/// 如果重载了DoExecute()且未调用base.DoExecute()，则本函数将不会执行到
```

```
    /// </summary>
    /// <param name="transaction">数据库事务</param>
        protected virtual void DoExecute(DbTransaction transaction) { }
```

按需二选一覆写即可。

23.2 示例

我们以一个简单的小功能，用途是让客户端获取到数据库的时间，来演示一下如何使用异步服务。

23.2.1 服务端

先看一下服务端代码：

```
  /// <summary>
  /// 数据库时间
  /// </summary>
  [Serializable]
  public class SysDateService : ServiceBase<SysDateService>
  {
    /// <summary>
    /// 处理执行指令
    /// 在运行持久层的程序域里被调用
    /// 请使用Phenix.Core.Data.DbCommandHelper与数据库交互以保证可扩展性
    /// 如果重载了DoExecute()且未调用base.DoExecute()，则本函数将不会执行到
    /// </summary>
    /// <param name="connection">数据库连接</param>
    protected override void DoExecute(DbConnection connection)
    {
        while (!ExecuteResult.Stop)
        {
          RefreshMessage(connection);
          Thread.Sleep(1000);
        }
    }

    private void RefreshMessage(DbConnection connection)
    {
      using (SafeDataReader reader = new SafeDataReader(connection,
@"select sysdate
  from PH_SystemInfo",
  CommandBehavior.SingleRow, false))
      {
        if (reader.Read())
        {
          ExecuteResult.Message = reader.GetDateTime(0);
        }
```

```
      }
    }
  }
}
```

黄底黑字为核心代码，while 循环体只要没有被告知 ExecuteResult.Stop（由客户端被 Phenix 传过来）将一直会执行下去。而希望被传回给客户端的内容，也就是数据库时间，应该被赋给 ExecuteResult.Message。

23.2.2 客户端

以下为客户端代码，可以看到客户端是如何向 ExecuteResult.Stop 赋值的，又是如何处理服务端传来的 ExecuteResult.Message：

```
new SysDateService().ExecuteAsync(DoProgressAsking, DoProgressAsked, askProgressInterval);


private static void DoProgressAsking(StopEventArgs args)
{
  Console.Write("是否继续向服务端发起询问?(Y/N)");
  if (String.Compare(Console.ReadLine(), "Y", StringComparison.OrdinalIgnoreCase) != 0)
    args.Stop = true;
}

private static void DoProgressAsked(ShallEventArgs args)
{
  Console.WriteLine("服务端返回消息对象: " + args.Message);
  if (args.Succeed)
    Console.WriteLine("结束");
    }
```

ExecuteResult.Message 可以传任何可序列化的对象，具体要看业务场景的需要。