CBF平台设计之批量应用框架

目录

[9. 批量应用框架 4](#_Toc504489308)

[9.1. 接口数据对象模型 6](#_Toc504489309)

[9.2. 环境资源信息访问 7](#_Toc504489310)

[9.3. 应用框架 8](#_Toc504489311)

[9.3.1. 全局静态对象 8](#_Toc504489312)

[9.3.2. 其他静态设计 10](#_Toc504489313)

[9.3.3. 框架对象 11](#_Toc504489314)

[9.4. 处理器和代理 14](#_Toc504489315)

[9.5. 调度器扩展 15](#_Toc504489316)

# 批量应用框架

批量应用框架（Batch Application Framework，简称BAF）模块以通信应用框架模块、基础服务模块、组件对象模块、数据访问框架和INTIGER平台为基础构建，致力于为批处理应用的开发、运行提供应用框架形式的完备的支持、约束，统一解决与业务无关的技术层面的问题。

#### 批量系统集群

基于批量应用框架搭建的批量系统可以部署成批量系统集群，这个集群通常由N个批量处理系统和1个批量调度系统等应用系统（逻辑节点）共同构成：



其中，批量调度系统（请注意！这个批量调度系统是一个逻辑层面的应用系统概念，物理层面，它由一个基于INTEGER调度引擎构建的批量调度服务器集群和加装在批量处理系统运行环境中的INTEGER代理共同构成）面向运行管理平台，处理其对整个批量系统集群发出的运行管理命令，并向其发送批量调度系统运行时产生的监控消息；批量处理系统面向批量调度系统，处理其提交的批处理请求。

#### 模块内部结构

批量应用框架模块由：1、接口数据对象模型；2、环境资源信息访问；3、应用框架；4、处理器和代理；5、调度器扩展；共5个子模块构成：

接口数据对象模型子模块确立了批处理应用接口数据对象体系的基本结构。

环境资源信息访问子模块提供了一组用以获取、设置批量系统集群运行时所需环境资源信息的DAO。

应用框架子模块确立了批量应用对象体系的基本结构，建立了应用功能实现的描述、登记和发现机制，在系统层面和具体批量请求处理层面提供了支持，使应用设计可以得到最大程度的简化。

处理器和代理子模块提供了通信层组件。处理器是批量处理系统中的通信层组件，用于解析批量调度系统提交的请求报文，调用应用框架接口进行处理，并组装应答报文回发给批量调度系统。代理是为批量调度系统提供的一组支持其基于接口数据对象进行访问的API。

调度器扩展子模块提供了一组扩展的INTIGER调度指令，通过代理对接处理器提交批量请求，并支持更加智能（例如动态任务分配）的批量调度流程设计。



#### 运行时子模块之间的协作

批处理时刻相关系统、模块之间主要的协作关系为：

1、运行管理平台访问批量调度系统，向其发出启动批处理命令；

2、批量调度系统调用环境资源信息访问子模块提供的接口获知批量系统集群运行时环境资源信息，然后即可通过处理器和代理子模块提供的代理访问批量处理系统等集群中所有节点，向它们发出启动命令；

3、批量处理系统上部署的处理器和代理子模块中的处理器调用应用框架子模块提供的接口，让其执行批量调度系统发来的启动命令；

4、批量处理系统上应用框架子模块调用环境资源信息访问子模块提供的接口获知批量处理系统运行时环境资源、配置等信息，执行启动命令相应的处理过程，完成批处理前的准备工作。

5、所有批量处理系统启动成功后，批量调度系统上的流程引擎执行批量调度流程，其调度器扩展子模块中的调度指令将基于接口数据对象模型子模块创建请求数据，并通过处理器和代理子模块提供的代理访问批量处理系统集群，向它们提交批处理请求。

6、批量处理系统上部署的处理器和代理子模块中的处理器解析批量调度系统发来的批处理请求报文，并基于接口数据对象模型子模块将其转换为请求数据，然后调用应用框架子模块提供的接口，让其处理批量调度系统发来的批处理请求；

7、批量处理系统上应用框架子模块调用应用提供的接口，让其执行批处理请求在应用层面的处理过程，并基于接口数据对象模型子模块产生应答数据；

8、批量处理系统上部署的处理器和代理子模块中的处理器基于接口数据对象模型子模块将应用框架子模块接口返回的应答数据转换成批处理应答报文，回发给批量调度系统。批量调度系统上处理器和代理子模块中的代理解析批量处理系统发来的批处理应答报文，将其转换为应答数据，返回给调度器扩展子模块中的调度指令处理，以便流程引擎继续执行调度流程。



## 接口数据对象模型

接口数据对象模型子模块确立了批处理应用接口数据对象体系的基本结构。

接口数据对象模型主要由应用标识（ApplicationIdentifier）、请求标识（RequestIdentifier）、请求文档（RequestDocument）、处理错误码（ProcessError）和应答文档（ResponseDocument）等数据构成。



#### 请求数据

客户端系统调用批量接口时必须发送以下请求数据：

1. 应用标识，是所需执行的某个应用功能的唯一标识。应用标识必须是8位长度由数字构成的字符串。应用标识的唯一性是相对于所访问的批量处理系统上的全部应用功能而言的——批量处理系统A上应用标识“00009001”所对应的应用功能，和批量处理系统B上应用标识“00009001”所对应的应用功能，并不一定是同一个。
2. 请求标识，是批量调度系统产生的某一次批量接口调用的唯一标识。请求标识由请求日期（RequestDate）和请求序号（RequestSequence）两部分内容构成。请求日期是批量调度系统认定的该次批量接口调用的处理日期，是符合“YYYY-MM-DD”格式的10位长度的日期字符串。请求序号则是相对于某一个请求日期内处理的所有批量接口调用，其中某一次调用的唯一标识。请求序号是14位长度由数字构成的字符串。
3. 请求文档，是一个RequestDocument派生类（最终基类是Component）对象，包含了由应用标识所标定的应用功能执行时所需输入的全部数据。请求文档包含的内容都是仅用来应对具体应用层面处理需要的，批量处理系统框架层面处理需要的应用标识、请求标识等等，都不是请求文档的内容。具体应用功能所需输入的请求文档类型是限定的。例如，ProcessAccount应用功能限定输入的请求文档必须是ProcessAccountReqestDocument类型，因此，批量调度系统调用批量接口时，如果其应用标识标定的是ProcessAccount应用功能，而请求文档并非ProcessAccountReqestDocument类型，则批量请求将不能得到正常的处理——需要注意的是，输入ProcessAccountReqestDocument派生类型的请求文档是可以的，此外，不同应用功能限定的请求文档类型有可能是同一个。

#### 应答数据

批量处理系统完成处理后，会向批量调度系统回送以下应答数据：

1. 处理错误码，表示批量请求是否得到正常处理。注意，这里的“正常”并不意味着“交易成功”，只要是获得了指定应用功能执行后产生的处理结果，均被视作正常，包括处理结果是“交易失败”的情况。处理错误码是一个整数，0表示正常，非0表示“系统异常”、...等异常。
2. 应答文档，是一个ResponseDocument派生类（最终基类是Component）对象，包含了由应用标识所标定的应用功能执行后会输出的全部数据。应答文档包含的内容都是仅用来表达具体应用层面处理结果的，批量处理系统框架层面处理产生的处理错误码等等，都不是应答文档的内容。具体应用功能在处理成功的情况下能够输出的应答文档类型是限定的。例如，ProcessAccount应用功能限定输出的应答文档是ProcessAccountResponseDocument类型，因此，批量调度系统调用批量接口时，如果其应用标识标定的是ProcessAccount应用功能，则处理成功情况下其能获得的应答文档只会是ProcessAccountResponseDocument类型——需要注意的是，不同应用功能限定的应答文档类型有可能是同一个。

应用处理失败时，都必须输出RejectResponseDocument类型（也是ResponseDocument派生类）的应答文档，应用可以设置其OperationHint属性值（字符串类型），表达失败的原因（通常是一个失败描述，或者是一个错误码），并向运行管理人员提示处理的方法。

## 环境资源信息访问

环境资源信息访问子模块提供了一组用以获取、设置批量系统集群运行时所需环境资源信息（例如，有哪些批量处理系统节点，等等）的环境资源信息访问对象类：批处理组件（BatchComponent）对象和批量调度（BatchScheduleComponent）对象，它们都是服务应用框架模块中定义的ServiceCenterComponent类型的衍生类对象，它们分别与批量系统集群中批处理系统、批量调度系统两类逻辑节点对应，描述具体逻辑节点基本设置。

## 应用框架

应用框架子模块确立了批处理应用对象体系的基本结构，建立了应用功能实现的描述、登记和发现机制。在系统级，为应用和配置的加载，运行时管控提供统一支持；在交易（具体批处理任务的处理）级，统一完成资源（数据库链接、事务、...）准备和释放、应用对象实例化和调度、日志记录（支持堵重）、错误/异常处理，并统一对外提供查询处理结果等接口，使应用设计可以得到最大程度的简化。

框架（Framework）类是应用框架子模块的核心设计。Framework类的设计可以分为静态和非静态两大部分，静态设计用以应对系统层面统一支持的需要，非静态设计则用以应对具体批量请求处理层面统一支持的需要。

* + 1. 全局静态对象

Framework类管理的最重要的全局静态对象包括：一个ComponentFactory对象，一个项目集合（ProjectCollection）对象和一个运行时设置（RuntimeSetting）对象。Framework类提供了相应的用以获取这些全局静态对象的静态属性（属性名称与访问属性获得对象的类名相同）。

Framework类是依据配置文件和与批量系统集群对应的ServiceCenter对象的相关属性内容来完成主要的初始化过程的。例如，基于组件工厂配置文件完成ComponentFactory全局静态对象初始化；依据ServiceCenter对象的BatchRuntimeSettingConfigName属性值和BatchProjectCollectionConfigName属性值获得相应的Config对象，并利用其中保存的组件对象——RuntimeSetting对象和ProjectCollection对象——来完成相应全局静态对象的初始化。

#### 组件工厂对象

ComponentFactory全局静态对象被用来代替new方法，完成批量处理系统运行时刻几乎所有组件对象的动态创建。接口数据被封装成数据组件类后，从请求报文转换成接口数据对象的过程，就会利用这个ComponentFactory对象，实现限定基类的数据对象的多态接收。此外，把应用内部的业务处理逻辑封装成应用组件类，就可利用这个ComponentFactory对象，完成应用组件对象的实例化，以便合理组织应用包。

#### 项目、功能实现和应用

应用框架子模块提供了Project类和Application类，共同构成了批量应用模型的基础。任何一个基于批量应用框架进行的业务处理功能相对完整的应用模块的开发工作，都需要定义一个项目类（Project派生类）和一组应用类（Application派生类），并实现其接口。

项目类最重要的接口是模块功能实现（ModuleImplementation）属性，该接口的实现要求返回一个模块功能实现（ModuleImplementation）对象，ModuleImplementation类是功能实现（Implementation）类的派生类，其内部又定义了一个功能实现对象集合成员，因此，可以把ModuleImplementation对象看作一棵Implementation对象树的中间节点。

Implementation类是纯虚类，可以实例化的衍生类，除了ModuleImplementation类，还有应用功能实现（ApplicationImplementation）类，后者的实例是Implementation对象树上更关键的、能够提供具体应用功能实现信息的对象——它具有所有Implementation对象都具有的Identifier（即应用标识）属性和Description属性，还有Application属性，访问Application属性即可获得执行相应应用功能的原型应用对象。

项目类ModuleImplementation（属性）接口的一般实现方式为：



#### 项目集合对象

ProjectCollection对象就是Project对象的集合。完成开发的应用模块，如果要加载到某个批量系统集群中运行，只需通过组件对象设计器把相关项目对象添加到相应ServiceCenter对象BatchProjectCollectionConfigName属性值所指向的Config对象内含的ProjectCollection对象中即可。

项目类是典型的组件类（最终基类是Component），ProjectCollection全局静态对象包含的所有Project对象在批量处理系统生命周期内又是一直存在的，因此，我们可以在项目类中定义一些可配置数据成员，作为应用模块内部具体批量请求处理时共用的参数——我们在通过组件对象设计器把Project对象添加到ProjectCollection对象时即可决定运行时批量处理系统中的由Project对象描述的参数（除了这些可配置数据成员外，项目类中还可以定义一些由相应应用模块自行管理的状态数据和环境资源对象成员，项目类可以提供相应的用以获取这些成员数据或对象的属性）。

为便于访问ProjectCollection全局静态对象包含的Project对象，批量处理系统启动时会创建一个以这些Project对象的Name属性值为键值的全局静态的Project对象哈希表，Framework类提供了GetProject静态方法，可以根据这些Project对象的Name属性值在该哈希表中查找到Project对象。

#### 应用注册器对象

Framework类管理了一个以应用标识为键值的存放应用注册器（ApplicationRegister）对象的全局静态哈希表，一个ApplicationRegister对象对应一个应用功能。ApplicationRegister对象内部管理了相应应用功能的实时流量统计值等数据，并关联了一个用以执行相应应用功能的原型应用对象。批量处理系统启动时会依据ProjectCollection全局静态对象包含的所有Project对象获得的ApplicationImplementation对象，完成ApplicationRegister对象全局静态哈希表的初始化。Framework类提供了GetApplicationRegister静态内部方法，可以根据应用标识在该哈希表中查找到ApplicationRegister对象。

#### 项目关联配置组件对象

在具体应用模块开发工作中，往往需要把项目的参数分成两个层面，一是批量处理系统完成初始化后即不再变化的参数，二是批量处理系统生命周期内会被运行管理操作变更的参数。我们通常用项目类的可配置数据成员满足第一个层面的需要，而为满足第二个层面的需要，我们要求每个Project对象都关联一个含有所需可配置数据成员的组件对象——我们称之为项目关联配置组件对象，批量处理系统运行时，可以调用这个对象的初始化和更新方法，更新其中的参数值，而不会对Project对象造成任何影响。

访问Project对象的ConfigedComponent属性可以获得相应的项目关联配置组件对象，ConfigedComponent属性是要求项目类实现的一个接口，因此，项目关联配置组件对象是什么类型，完全由项目类的实现决定——在应用模块开发工作中，对应每个项目类，几乎都会定义一个项目关联配置组件对象类。批量处理系统启动时也会创建一个以Project对象的Name属性值为键值的存放项目关联配置组件对象的全局静态哈希表，Framework类提供了GetConfigedComponent静态方法，可以根据Project对象的Name属性值在此哈希表中查找到相应的项目关联配置组件对象。

#### 运行时设置对象

项目关联配置组件对象可通过RuntimeSetting全局静态对象进行初始化。RuntimeSetting对象内部聚合了一个项目运行时设置（ProjectRuntimeSetting）对象集合对象，其中包含的每个ProjectRuntimeSetting对象内部又聚合了一个项目关联配置组件对象。批量系统启动时刻，会获取RuntimeSetting全局静态对象聚合的全部项目关联配置组件对象，用其作为原型，初始化Project对象全局静态哈希表里的Project对象所关联的项目关联配置组件对象——我们在通过组件对象设计器设计Config对象中的RuntimeSetting对象时即可决定运行时批量系统中的由项目关联配置组件对象描述的参数。

应用类也是典型的组件类（最终基类是Component），因此，我们可以在应用类中定义一些可配置数据成员，作为应用内部处理时的参数。RuntimeSetting对象内部聚合了一个应用运行时设置（ApplicationRuntimeSetting）对象集合对象，其中包含的每个ApplicationRuntimeSetting对象内部又聚合了一个原型应用对象。批量系统启动时刻，会获取RuntimeSetting全局静态对象聚合的全部原型应用对象，用其作为原型，初始化ApplicationRegister对象全局静态哈希表里的ApplicationRegister对象所引用的原型应用对象——我们在通过组件对象设计器设计Config对象中的RuntimeSetting对象时即可决定运行时批量系统中的由应用对象描述的参数。

* + 1. 其他静态设计

除了上述全局静态对象，Framework类还管理了一组批量处理系统在从启动到关停整个生命周期内都需要使用的静态的状态数据和环境资源对象，并提供了相应的用以获取这些对象的静态属性（访问属性获得对象的类名基本都与属性名称相同），包括：系统状态（IsReady属性），逻辑节点对应的环境资源信息（BatchProcessComponent属性和ServiceCenter属性），访问服务处理系统集群的代理（ServiceProxy属性），等等。

Framework类还封装了一组静态方法，用以完成批量处理系统启动时的处理过程（StartBatch方法），系统关闭时的处理过程（StopBatch方法），以及系统运行过程中的查询（QueryState方法、...等）等运行管理操作的处理过程。

* + 1. 框架对象

批量系统在处理一个具体批量请求时，需实例化一个Framework对象，并调用Framework对象提供的批量请求处理方法——这个非静态的Framework对象其生命周期与具体批量请求在应用框架中的整个处理过程对应。

Framework对象的内部，管理了一组与具体批量请求处理对应的状态数据和环境资源对象，包括：所有批量接口数据，数据库链接对象，等等，访问Framework对象的属性（ApplicationIdentifier属性、RequestIdentifier属性、EnterpriseResourceSqlService属性、...等）即可获得这些数据或对象。

#### InstanceFramework属性

为了简化应用的编写，Framework对象并不作为参数传递给应用对象，而是由Framework类提供了一个静态属性（InstanceFramework），可以让应用随时获得Framework对象。

#### BeginProcess/EndProcess方法

Framework对象提供的批量请求处理方法是BeginProcess/EndProcess。

BeginProcess/EndProcess方法也是遵照微软提出的异步应用编程模型设计的，目的是能够告知调用者批量请求处理过程的进展状况，而不只是最终完成的结果。

Framework对象提供的BeginProcess方法的原型为：



BeginProcess方法的主要处理过程如下：

1、把当前Framework对象放入静态ThreadLocal对象，以便后续设计访问Framework类的InstanceFramework静态属性获得当前Framework对象。

2、创建一个被用作BeginProcess方法返回值的异步处理状态对象，并初始化Framework对象中的相应变量，以便后续设计能够不断更新这个异步处理状态对象。

3、依据BeginProcess方法参数中的请求数据初始化Framework对象中的相应变量。

4、记录与“进入”事件对应的批量日志（日志数据库中的BatchJournal表）。

5、依据应用标识检索到相应的ApplicationRegister对象，并以ApplicationRegister对象关联的应用对象为原型，克隆一个用来执行具体处理过程的应用对象（我们也称之为工作应用对象）。

6、创建业务数据库链接对象，初始化Framework对象中的相应变量，以便后续设计访问Framework对象的EnterpriseResourceSqlService属性获得业务数据库链接对象，并把该数据库链接对象设置为数据访问框架模块中的默认数据库链接对象。

7、以请求文档为参数调用工作应用对象的Process方法，把其返回的应答文档放入待返回的异步处理状态对象。

8、关闭业务数据库链接对象，调用Complete方法并返回异步处理状态对象——调用者获得该异步处理状态对象后即可知道批量请求处理过程已完成，调用者随后即可调用Framework对象的EndProcess方法来获得具体批量请求的处理结果。

BeginProcess方法中调用的Complete方法的原型为：



这个Complete方法执行时，静态ThreadLocal对象中的Framework对象引用将被清除，继而把处理错误码（参数nProcessError）放入待返回的异步处理状态对象，并设置异步处理状态对象的IsCompletedSynchronously属性值为true。

Framework对象提供的EndProcess方法的原型为：



EndProcess方法的主要处理过程如下：

1、记录与“返回”事件对应的批量日志（日志数据库中的BatchJournal表）。

2、使用Framework对象和先前返回的异步处理状态对象中的相应变量，设置EndProcess方法参数中的应答数据。

#### 应用对象的Process方法和OnProcess接口

与Framework对象协作完成整个批量请求处理过程的，是工作应用对象。Framework对象会调用工作应用对象的Process方法（这是一个内部方法），Process方法则会调用该应用对象的OnProcess接口来完成具体批量请求在应用层面的处理过程。

应用对象的OnProcess接口的原型为：



应用在实现OnProcess接口时，首先应当创建一个在完成所有预期处理过程后才会返回的应答文档。其后，可以访问Framework对象的EnterpriseResourceSqlService属性获得业务数据库链接对象，执行所需的数据库操作，并在处理过程中依据数据库操作结果设置应答文档中的数据。如果由于某些业务规则不能满足，而无法完成所有预期的处理过程，则需立即创建一个RejectResponseDocument对象，设置其OperationHint属性值并返回。

#### 异常处理

OnProcess接口实现的处理过程通常无需自行捕捉异常。当异常发生时，应用对象的Process方法会统一捕捉异常，完成异常信息的记录（日志数据库中的BatchException表），并返回一个OperationHint属性值被设置为"系统异常，请联系技术支持中心"的RejectResponseDocument对象。但这并不是绝对的。某些异常应当依据业务设计将其理解为某个业务规则不能满足，或者进入某个处理过程分支的先决条件。还有一些异常需要进行异常描述信息的转换，更加清晰地指出是哪一个处理环节发生了错误，即创建一个以原异常作为内部异常的新异常，设置新异常的描述信息并立即抛出。

#### 日志记录

应用框架子模块在具体批量请求的处理过程中会自行记录主要环节的日志（日志数据库中的BatchJournal表），例如，进入应用层处理时的应用标识、时间、请求文档，从应用层处理返回时的时间、应答文档等等，因此，应用在OnProcess接口实现的处理过程中，通常无需再自行记录日志。但在开发和试运行阶段，为了方便查找问题，应用仍然有在OnProcess接口实现的处理过程中记录日志的需要。Framework对象提供了SaveJournal方法用以支持应用记录日志（日志数据库中的BatchJournal表）——实际上，应用框架子模块自己在记录日志时也是调用SaveJournal方法。SaveJournal方法的原型为：



#### 批量登记簿

具体批量请求的处理过程的相关信息将会被记入批量登记簿（日志数据库中的BatchRegister表），包括：应用标识，请求标识，应答标识，请求文档和应答文档等域。这一动作由Framework对象的BeginProcess方法在创建工作应用对象后执行。

批量登记簿基于请求标识域定义了唯一索引——这意味着如果批量调度系统调用数据加工批量接口时提交的请求标识重复，则会引发批量登记簿插入异常，立即调用Complete方法并返回异步处理状态对象，处理过程结束，批量调度系统将会得到“请求标识重复”处理错误码。

批量调度系统调用批量接口时可能会由于通信故障，导致其未能在指定时间内收到应答，此时，批量调度系统通常都有查明处理结果的需要。基于批量登记簿，Framework对象的BeginProcess方法就可以满足这个需要：调用BeginProcess方法时，把第一个参数bAllowRepeat设置为true，如果发生批量登记簿插入重复的异常，BeginProcess方法就会从批量登记簿中找出请求标识相同的那条记录，并返回其中的应答标识和应答文档，如果没有发生插入重复异常，则按照BeginProcess的正常逻辑进行后续处理。

在执行批量登记簿插入操作时，应答文档域只能填充null，处理状态域将被填充为"P"——正在处理中。Framework对象的BeginProcess方法完成应用对象的Process方法调用后，会依据其返回的应答文档更新先前插入的批量登记簿记录的应答文档、处理状态等域。如果返回的应答文档是一个RejectResponseDocument对象，则处理状态域将被填充为"R"——批量请求被拒绝，否则，处理状态域将被填充为"A"——批量请求被接受。

在处理状态域尚为"P"时，如果批量调度系统提交查明处理结果的请求，只能得到“原批量请求还在处理中”处理错误码。

## 处理器和代理

处理器和代理子模块提供了通信层的组件，在批量处理系统端负责解析批量调度系统提交的请求报文，调用应用框架接口进行处理，并组装应答报文回发给批量调度系统——即所谓的批量处理器。

为进一步简化批量调度系统访问批量处理系统时的编程，处理器和代理子模块还为批量调度系统提供了一组API，这些API自动建立、维护通信通道，并自动完成符合处理器访问报文格式要求的请求报文的组装和应答报文的解析——即所谓的批处理代理。

批量处理器实际上是两个，一是基于通信应用框架模块的异步基础服务应用模型开发的（AsynchronousServiceApplication派生类AsynchronousServiceProcessor），所登记的基础服务标识为"ABCS"，调用Framework对象提供的BeginProcess.../EndProcess...等异步API，处理批量调度系统提交的具体批量请求；二是基于通信应用框架模块的基础服务应用模型开发的（ServiceApplication派生类ServiceProcessor），所登记的基础服务标识为"SBCS"，调用Framework类/对象提供的Query.../Start.../Stop...等同步API，处理批量调度系统提交的启动、关闭、状态查询等命令。

批量调度系统基于基础服务访问协议对批量处理器进行访问——在基础服务访问请求报文应用区中填写批量处理器能够执行的处理方法名称和相关输入参数，并在基础服务访问应答报文应用区中读取批量处理器的处理结果和其他输出参数。应用区报文格式和服务处理系统中的服务处理器应用区报文格式相同。

AsynchronousBatchProcessor支持Process处理方法，BatchProcessor支持QueryState、StopBatch、StartBatch等处理方法，这些处理方法的作用和相应的其他要素的含义、类型和填写格式如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 处理方法名 | 作用 | 请求报文其他要素 | 应答报文要素 |
| Process |  |  |  |

批处理代理（ServiceProxy）对应批量处理器支持的处理方法，提供以下方法：



## 调度器扩展

调度器扩展子模块提供了一组扩展的INTIGER调度指令，通过代理对接处理器提交批处理任务，并支持更加智能（例如动态任务分配）的批量调度流程设计。