ParaView Catalyst API

李健

Catalyst API

下面介绍适配器如何将信息在模型代码与Catalyst之间传递信息。

信息可分解为3个部分：

1.VTK数据对象 (Data Objects)

2.管线 (Pipeline)

3.控制信息 (Control Information)

VTK数据对象：包含管线的输入的信息。

管线定义对数据执行何种操作以及如何输出结果。

为了管线可以合适地执行，控制信息指定各管线何时执行，以及VTK数据对象需要的何种信息。

Catalyst API

High-Level View

- 第一步是初始化，设置Catalyst环境，创建之后要执行的管线。这一般在执行完MPI_init( )之后立即执行。
- 第二步是执行管线（如果需要）。这步一般在各时间步更新后执行。
- 最后一步是结束Catalyst，一般在执行MPI_Finalize( )之前执行。

第一步和最后一步很简单，但中间一步比较复杂。原则上，中间步要查询，看看是否要求执行。如果不要执行，则立即返回控制到模型代码。如果一个或多个管线需要重复执行，则适配器需要更新网格描述和属性信息的VTK数据对象，然后执行要求的管线。根据管线中过滤器需要完成的工作量，这要花费的时间长短不一。当所有需要执行的管线完成后，控制返回到模型代码。

Catalyst API

Class(类) API

主要的Catalyst API类有：

- vtkCPProcessor
- vtkCPDataDescription
- vtkCPInputDataDescription
- vtkCPPipeline
- 为Python定义的衍生类

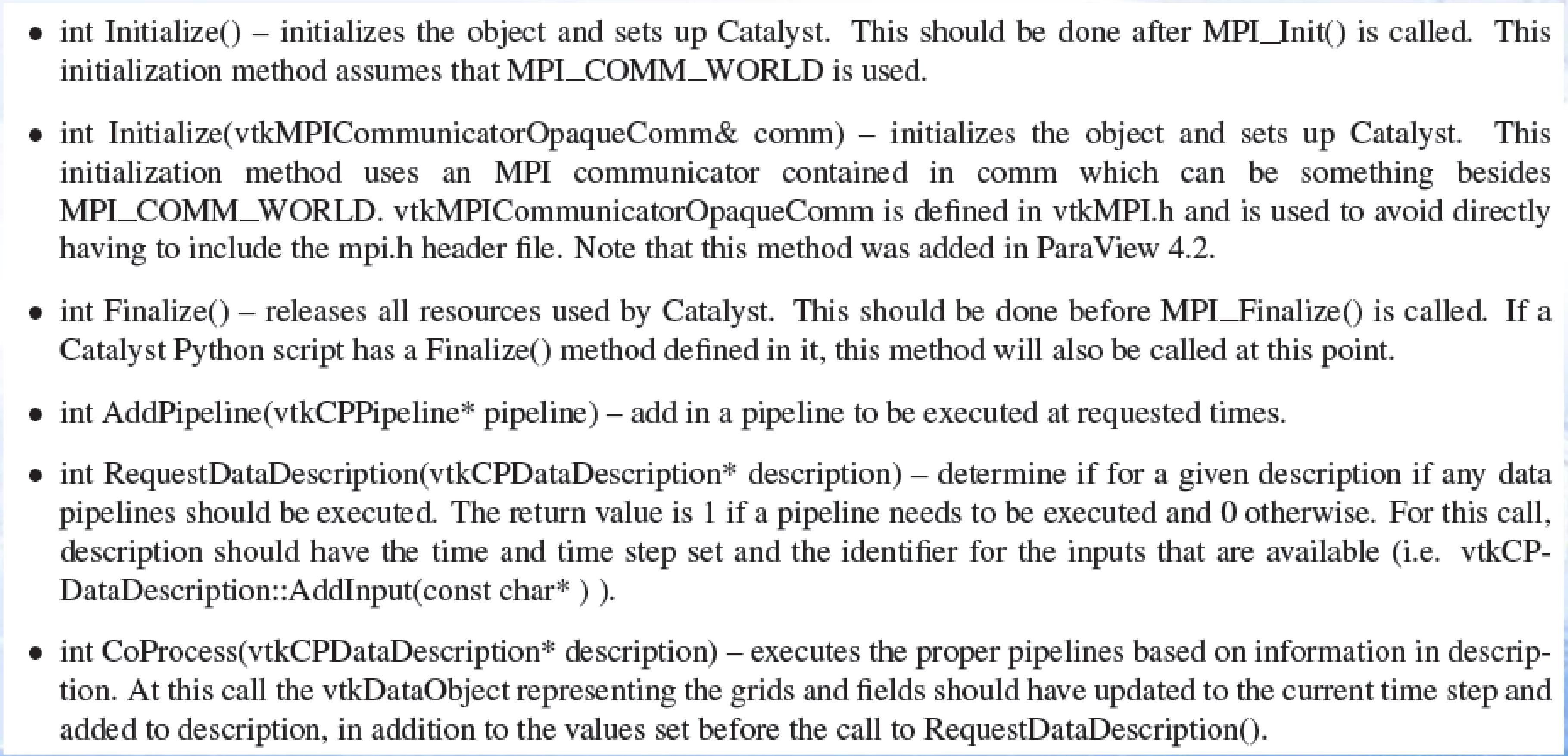
当Catalyst编译时支持Python，所有类都做Python封装。

Catalyst API

- vtkCPProcessor

vtkCPProcessor负责管理管线。包括：存储管线、查询管线看是否需要执行。

vtkCPProcessor的方法有：



- int Initialize() – initializes the object and sets up Catalyst. This should be done after MPI_Init() is called. This initialization method assumes that MPI_COMM_WORLD is used.
- int Initialize(vtkMPICommunicatorOpaqueComm& comm) – initializes the object and sets up Catalyst. This initialization method uses an MPI communicator contained in comm which can be something besides MPI_COMM_WORLD. vtkMPICommunicatorOpaqueComm is defined in vtkMPI.h and is used to avoid directly having to include the mpi.h header file. Note that this method was added in ParaView 4.2.
- int Finalize() – releases all resources used by Catalyst. This should be done before MPI_Finalize() is called. If a Catalyst Python script has a Finalize() method defined in it, this method will also be called at this point.
- int AddPipeline(vtkCPPipeline* pipeline) – add in a pipeline to be executed at requested times.
- int RequestDataDescription(vtkCPDataDescription* description) – determine if for a given description if any data pipelines should be executed. The return value is 1 if a pipeline needs to be executed and 0 otherwise. For this call, description should have the time and time step set and the identifier for the inputs that are available (i.e. vtkCP-DataDescription::AddInput(const char*)).
- int CoProcess(vtkCPDataDescription* description) – executes the proper pipelines based on information in description. At this call the vtkDataObject representing the grids and fields should have updated to the current time step and added to description, in addition to the values set before the call to RequestDataDescription().

Catalyst API

● vtkCPProcessor

上述方法提供的基本功能足够了。如果adaptor代码需要在模拟运行时手动删除管线，可使用如下方法：

•int GetNumberOfPipelines() – get the number of existing pipelines.

•vtkCPPipeline* GetPipeline(int i) – get the ith pipeline.

•void RemoveAllPipelines() – remove all of the existing pipelines from the co-processor and prevents them from being executed.

•void RemovePipeline(vtkCPPipeline* pipeline) – removes a pipeline and prevents it from being executed.

Catalyst API　　vtkCPPipeline and vtkCPPythonScriptPipeline

vtkCPPipeline是存储将要由vtkCPProcessor执行的VTK管线的抽象基础类。

vtkCPPythonScriptPipeline是实际衍生类，Python脚本执行，用于存储VTK管线。

vtkCPPipeline的方法有：

- int RequestDataDescription(vtkCPDataDescription *description) – given description, return 1 if this pipeline needs to execute and 0 otherwise.
- int CoProcess(vtkCPDataDescription *description) – execute the pipeline stored by this object and return 1 for success and 0 for failure.

Catalyst API vtkCPPipeline and vtkCPPythonScriptPipeline

注意：vtkCPPipeline中的方法都是抽象的。对于C++代码的管线，希望用户在一个由vtkCPPipeline衍生的类中创建。

Python管线在vtkCPPythonScriptPipeline中实施。

对于vtkCPPythonScriptPipeline，仅有的方法是：

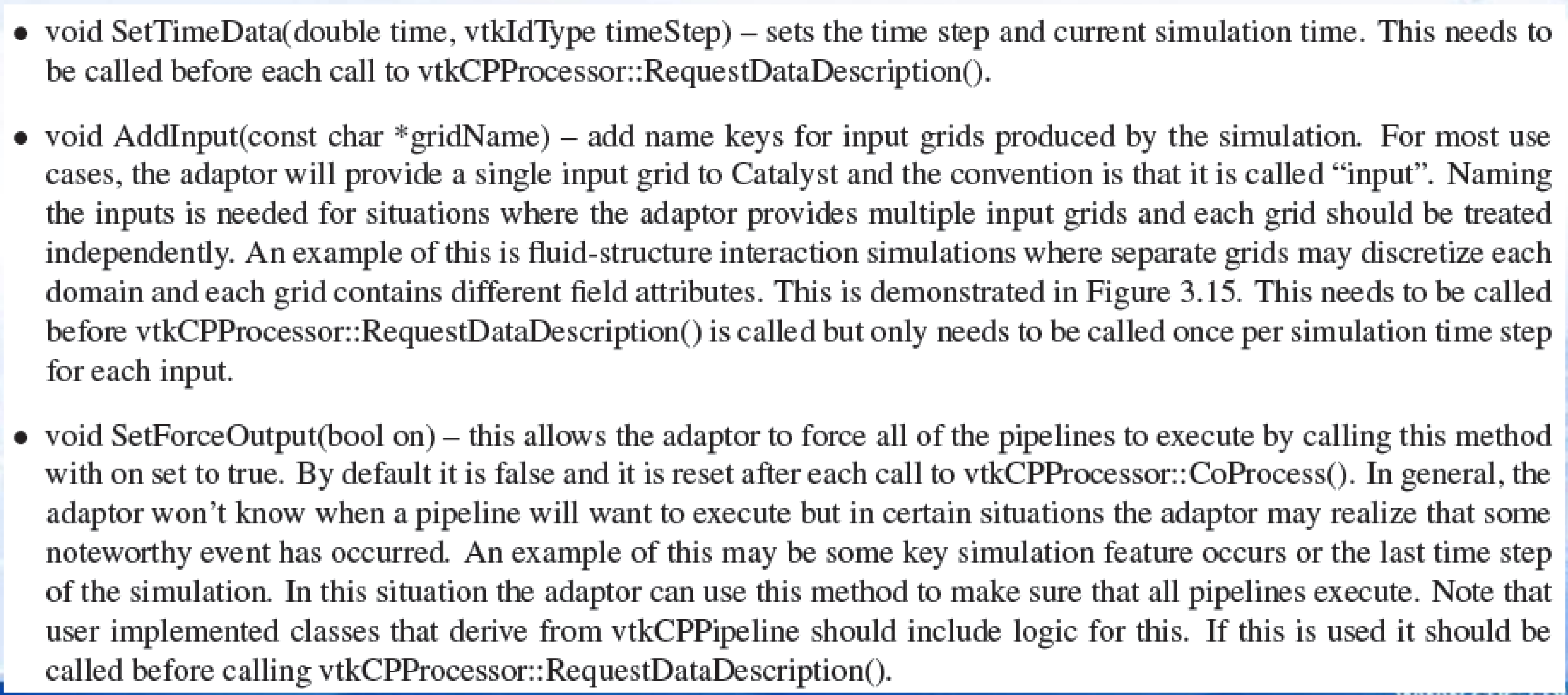
- int Initialize (const char *fileName) – initialize the pipeline with the file name of a Python script.

Catalyst API vtkCPDataDescription

vtkCPDataDescription类，用于的存储在adaptor与管线之间传递的信息。

Adaptor提供一些信息，管线使用，管线提供另一部分信息，adaptor使用。

Adaptor提供给管线的信息的方法有：



- void SetTimeData(double time, vtkIdType timeStep) – sets the time step and current simulation time. This needs to be called before each call to vtkCPProcessor::RequestDataDescription().
- void AddInput(const char *gridName) – add name keys for input grids produced by the simulation. For most use cases, the adaptor will provide a single input grid to Catalyst and the convention is that it is called “input”. Naming the inputs is needed for situations where the adaptor provides multiple input grids and each grid should be treated independently. An example of this is fluid-structure interaction simulations where separate grids may discretize each domain and each grid contains different field attributes. This is demonstrated in Figure 3.15. This needs to be called before vtkCPProcessor::RequestDataDescription() is called but only needs to be called once per simulation time step for each input.
- void SetForceOutput(bool on) – this allows the adaptor to force all of the pipelines to execute by calling this method with on set to true. By default it is false and it is reset after each call to vtkCPProcessor::CoProcess(). In general, the adaptor won’t know when a pipeline will want to execute but in certain situations the adaptor may realize that some noteworthy event has occurred. An example of this may be some key simulation feature occurs or the last time step of the simulation. In this situation the adaptor can use this method to make sure that all pipelines execute. Note that user implemented classes that derive from vtkCPPipeline should include logic for this. If this is used it should be called before calling vtkCPProcessor::RequestDataDescription().

Catalyst API

vtkCPDataDescription

提供单个管线输入的adaptor，例如仅调用AddInput()一次，上述2中方法的形参分别是0和”input”。

如果adaptor提供多个网格输入，可能不需要所有形参。

为确定需要哪种方式来更新要求的管线，可使用如下方法：

- bool GetIfGridIsNecessary(const char* name)

Catalyst API

vtkCPDataDescription

当需要vtkCPDataDescription传递以上信息，在adaptor与pipeline之间往返，对于用户开发的管线，可能需要传递更多往返信息。这种情况下，需要使用一个用户数据对象。目前，我们使用vtkFieldData对象实现该功能。

Python封装通过使用从vtkAbstractArray衍生的类（也Python封装），可使用多种数据类型。方法有：

- void SetUserData(vtkFieldData* data)
- vtkFieldData* GetUserData()

Catalyst API

● vtkCPInputDataDescription

vtkCPInputDataDescription类，与vtkCPDataDescription类似，负责传递adaptor与pipeline之间的信息。区别是：vtkCPInputDataDescription传递有关网格(grids)和场(fields)的信息。

应该在vtkCPDataDescription（为adaptor提供的各单独的输入VTK数据对象）内有1个vtkCPInputDataDescription对象。主要方法有：

- void SetGrid (vtkDataObject *grid) – set the input data object representing the grids and their attributes for the pipelines.
- void SetWholeExtent (int, int, int, int, int, int) or void SetWholeExtent (int[6]) – for topologically regular grids, set the whole extent for the entire grid.

还有其他可提高adaptor效率的方法。目的是：提醒adaptor代码，管线需要何种属性。

适配器：把所有的放一起

下面介绍在适配器中如何将上述的类整合在一起。

1、初始化步骤：

（1）创建vtkCPProcessor对象，调用Initialize( )；

（2）创建vtkCPPipeline对象，将他们添加到vtkCPProcessor。

适配器：把所有的放一起

2、调用**co-processing**子程序：

（1）创建vtkCPDataDescription对象(*)；

1)调用SetTimeData();

2)对每一个输入数据对象，调用AddInput()，使用关键标识字符串(*)；

3)选择性地，调用SetForceOutput()。

（2）调用vtkCPProcessor::RequestDataDescription()，使用创建的vtkCPDataDescription对象；

1)如果RequestDataDescription()返回0，则返回控制到模型代码；

2)如果RequestDataDescription()返回1，则：

2.1：对每一个vtkCPInputDataDescription创建vtkDataObject和属性，添加他们，使用vtkCPDataDescription::GetInputDataDescriptionByName(const char* name)->SetGrid()

2.2：调用vtkCPProcessor::CoProcess()

适配器：把所有的放一起

（3）结束步：调用vtkCPProcessor::Finalize()，并删除vtkCPProcessor对象。

注意：带星号(*)的项目，在首次执行co-processing子程序后完成，之后保持不变的数据结构。

下面通过一个示例代码说明一个简单的适配器程序：

```
// declare some static variables
vtkCPProcessor* Processor = NULL;
vtkUnstructuredGrid* VTKGrid;

// Initialize Catalyst and pass in some file names
// for Python scripts.
void Initialize(int numScripts, char* scripts[])
{
  if(Processor == NULL)
    {
    Processor = vtkCPProcessor::New();
    Processor->Initialize();
    }
  else
    {
    Processor->RemoveAllPipelines();
    }
  // Add in the Python script
  for(int i=1;i<numScripts;i++)
    {
    vtkCPPythonScriptPipeline* pipeline =
     vtkCPPythonScriptPipeline::New();
    pipeline->Initialize(scripts[i]);
    Processor->AddPipeline(pipeline);
    pipeline->Delete();
    }
}

// clean up at the end
void Finalize()
{
  if(Processor)
    {
    Processor->Delete();
    Processor = NULL;
    }
  if(VTKGrid)
    {
    VTKGrid->Delete();
    VTKGrid = NULL;
    }
}

// The simulation calls this method at the end of every time
// step. grid and attributes are the simulation data structures.
// lastTimeStep is a flag indicating whether this will be
// the last time CoProcess is called. It will force all of
// the pipelines to execute.
void CoProcess(Grid& grid, Attributes& attributes, double time,
               unsigned int timeStep, bool lastTimeStep)
{
  vtkCPDataDescription* dataDescription =
    vtkCPDataDescription::New();
  // specify the simulation time and time step for Catalyst
  dataDescription->AddInput("input");
   dataDescription->SetTimeData(time, timeStep);
  if(lastTimeStep == true)
    {
    // assume that we want to all the pipelines to execute if it
    // is the last time step.
    dataDescription->ForceOutputOn();
    }
  if(Processor->RequestDataDescription(dataDescription) != 0)
    {
    // Catalyst wants to perform co-processing. We need to build
    // the VTK grid and set the attribute information on it now.
    BuildVTKDataStructures(grid, attributes);
    // Make a map from "input" to our VTK grid so that
    // Catalyst gets the proper input data set for the pipeline.
    dataDescription->GetInputDescriptionByName("input")->
      SetGrid(VTKGrid);
    // Call Catalyst to execute the desired pipelines.
    Processor->CoProcess(dataDescription);
    }
  dataDescription->Delete();
}
```