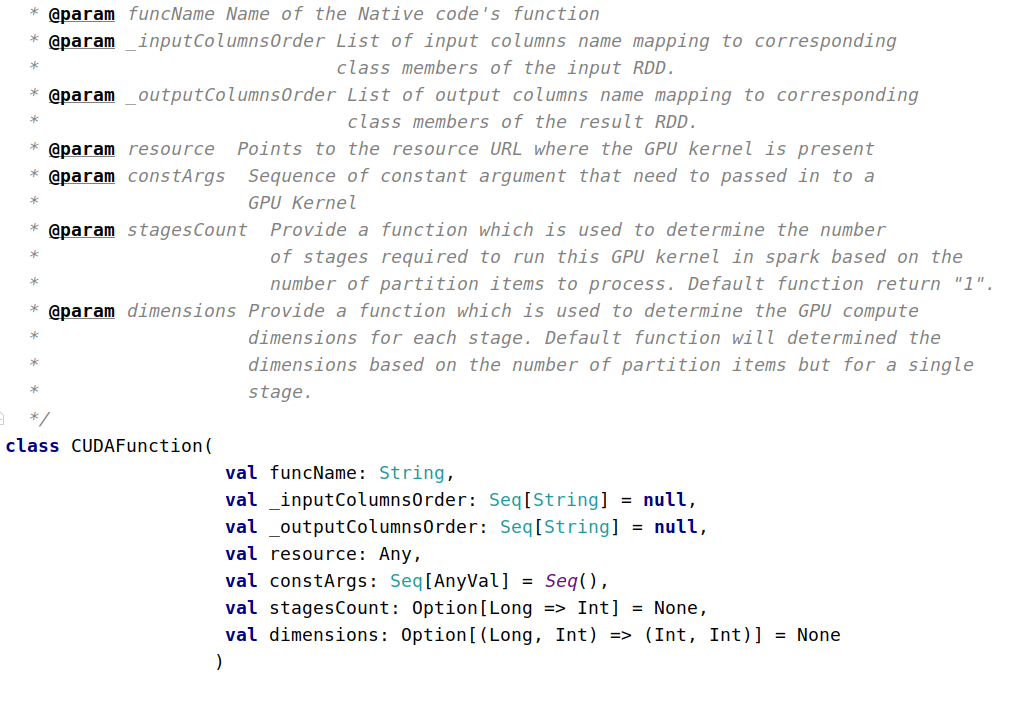
# 普通RDD使用

## CUDAFunction

这个class用于描述一个kernel函数。在一个RDD要执行GPU操作时，作为参数传给mapExtFunc和reduceExtFunc。

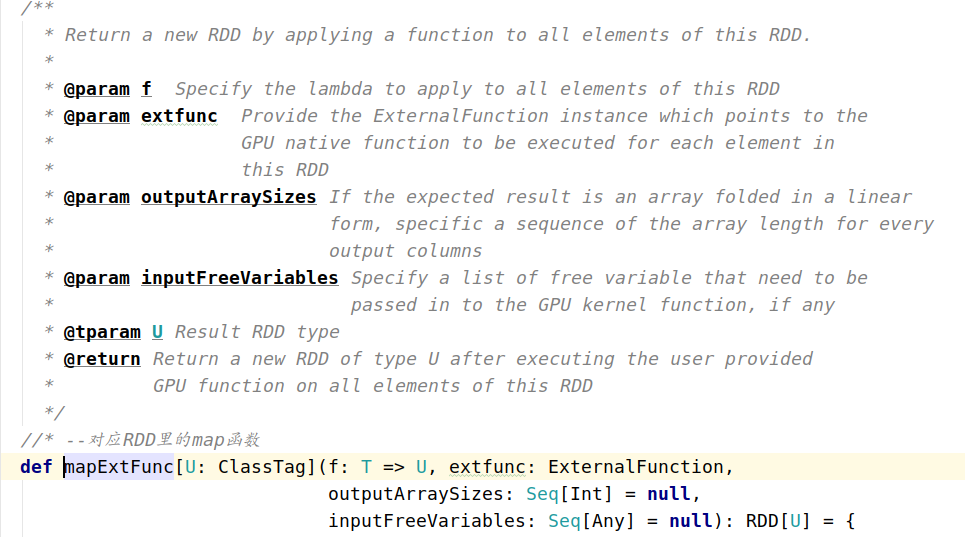


参数说明：

|  |  |
| --- | --- |
| funcName | 核函数的函数名 |
| \_inputColumnsOrder | Dataset中，作为核函数输入的类的成员名字 |
| \_outputColumnsOrder | Dataset中，作为接收核函数返回结果的类的成员名字 |
| resource | 读取的Ptx文件，URL形式 |
| constArgs | Constant arguments，cuda核函数用到的const参数。（例如逻辑回归中，向量的维度，初始权重） |
| stagesCount | 是一个(size:Long=>totalStages:Int)的函数，根据数据元素的数量(size)来决定次kernel执行的stages。目前还没明白为什么Kernel要执行多个stage。 |
| dimensions | 核函数的维度，即gridSize.x和blockSize.x |

这里的dimensions和dataset的gpuParams参数不同，只能指定一维的grid和block，也不能指定共享存储的大小。函数形式应该是（size:Long，stage:Int）=>(gridSize:Int，blockSize:Int)。此处后面应该可以修改一下支持使用三维。

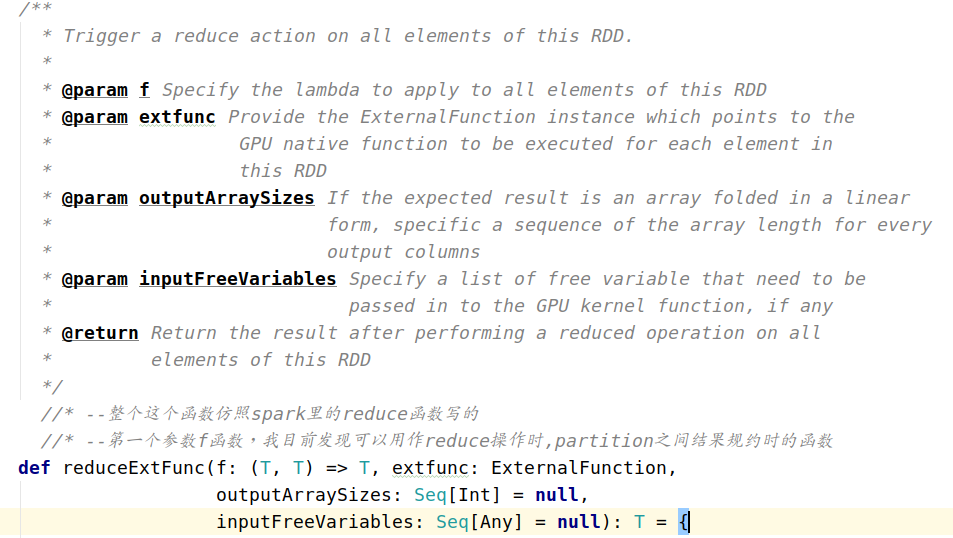
## 二．mapExtFunc



参数说明

|  |  |
| --- | --- |
| f | 指定对这个RDD中所有elements的操作的表达式，  mapExtFunc中的第一个参数，是一个CPU函数。如果工作程序节点没有GPU，它将作为备用函数，在CPU上使用 |
| extfunc | 包装成DSCUDAFunction的kernel函数 |
| outputArraySizes | 如果预期的返回结果是一个包装成线性形式的数组，为每个输出列指定结果的数组长度。即seq里的每个元素对应每个输出数组的长度。 |
| inputFreeVariables | 需要传播的 broadcast data？？ |

## 三．reduceExtFunc

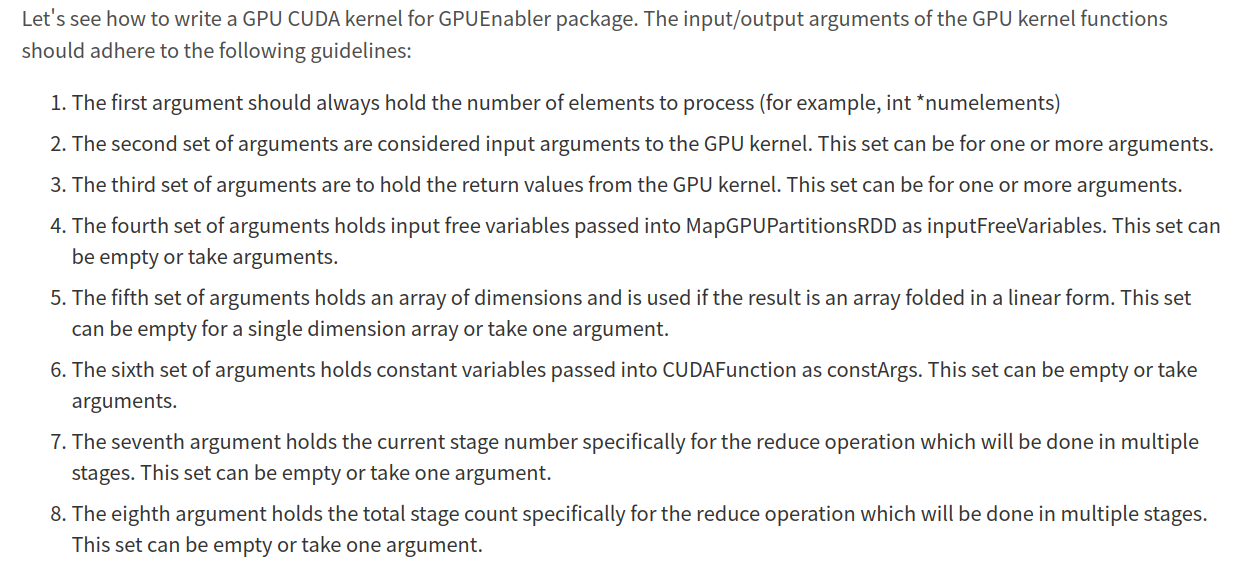


参数说明

|  |  |
| --- | --- |
| f | 作为reduce操作规约用的函数，对每个Partion的结果进行操作 |
| extfunc | 包装成CUDAFunction的kernel函数 |
| outputArraySizes | 如果预期的返回结果是一个包装成线性形式的数组，为每个输出列指定数组长度的序列 |
| inputFreeVariables | 需要传播的 broadcast data |

## 四．Cuda函数编写

<https://blog.csdn.net/weixin_33813128/article/details/92209982>这个blog中简单介绍了如何编写对应的核函数



cuda核函数的参数列表有如此要求：

1.第一个参数是要处理的element的个数，例如int \* numelements。从CUDAFunction.scala文件中，compute函数内对kp(kernel parameters)的定义可以看到，强制第一个参数为size，不过在写CUDAFuncion时不用考虑，gpuenabler在调自动生成这个参数。

2.第二组参数被视为GPU kernel的输入参数，可以一个或多个参数。对应CUDAFuncion里的inputColumnsOrder。

3.第三组参数用于保存GPU kernel的返回值，可以一个或多个参数。对应CUDAFuncion里的outputColumnsOrder。

4.第四组参数，是作为输入FreeVariables传递到MapGPUPartitionsRDD的无输入变量。此集合可以为空或接受参数？？？?没看懂和MapGPUPartitionsRDD有什么关系。对应mapExtFunc和reduceExtFunc里的inputFreeVariables。

5.第五组参数保存维数组，如果结果是线性折叠的数组，则使用第五组参数。对于单个维数组，此集可以为空，也可以采用一个参数。对应mapExtFunc和reduceExtFunc里的outputArraySizes。

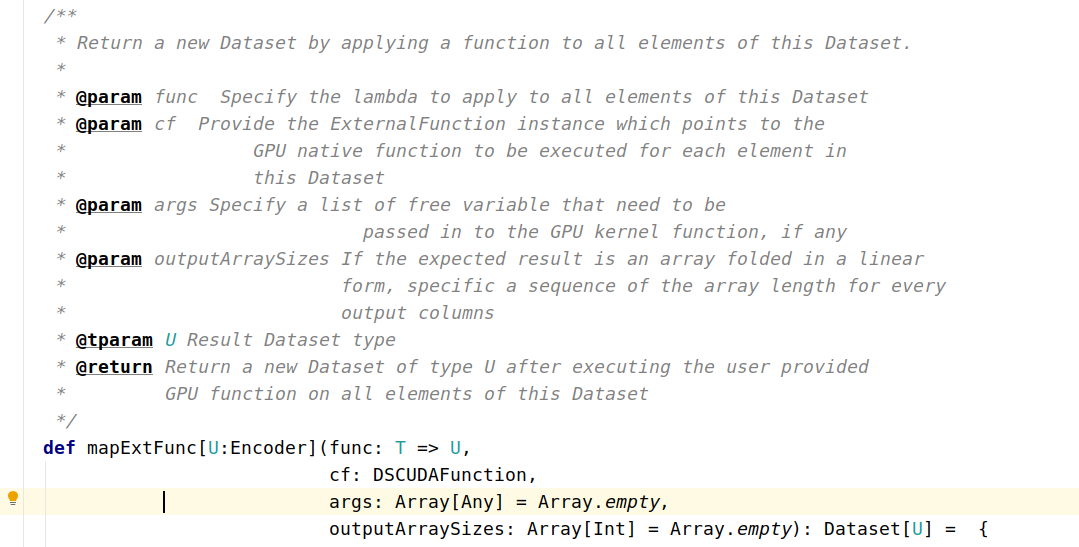
6.第六组参数保存作为constArgs传递到CUDAFunction的常量变量。该集合可以为空或接受参数。对应CUDAFuncion里的constArgs。

7.第七个参数保存当前阶段号，该阶段号专门用于将在多个阶段中进行的reduce操作。该集合可以为空，也可以采用一个参数。作用还未知。这个参数根据CUDAFuncion里的stagesCount函数参数生成其值。stagesCount根据rdd中数据量的大小计算出kernel要执行的总次数totalStages，然后从0 -> totalStages-1

8.第八个参数保存专门用于reduce操作的阶段总数，该操作将分多个阶段进行。该集合可以为空，也可以采用一个参数。根据CUDAFunction.scala中对kp(kernel parameter)的逻辑，目前看来如果spark代码中传递了这个参数，则kernel函数中这个和第7个参数都要实现。这个参数也根据CUDAFuncion里的stagesCount生成其值。

# Dataset使用

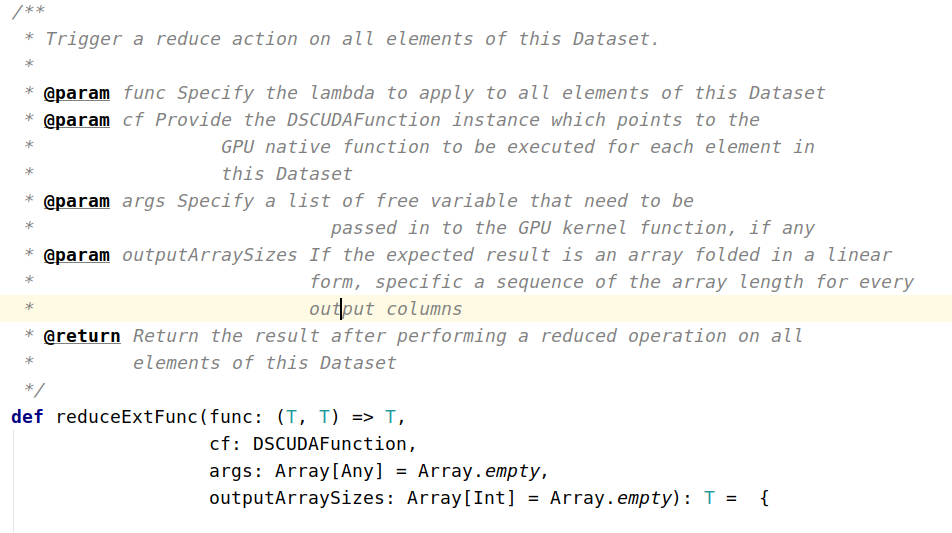
## Dataset.mapExtFunc



参数说明

|  |  |
| --- | --- |
| func | 指定对这个dataset中所有elements的操作的表达式，  mapExtFunc中的第一个参数，是一个CPU函数，如果工作程序节点没有GPU，它将作为备用函数 |
| cf | 包装成DSCUDAFunction的kernel函数 |
| args | 需要传给GPU核函数的一系列自由变量 |
| outputArraySizes | 如果预期的返回结果是一个包装成线性形式的数组，为每个输出列指定结果的数组长度 |

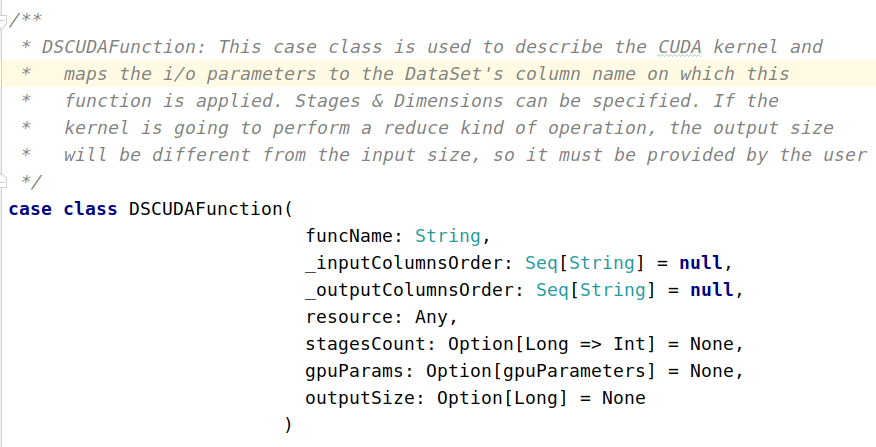
## Dataset.reduceExtFunc



参数说明

|  |  |
| --- | --- |
| func | 另外推测还作为reduce操作规约用的函数，对每个Partion的结果进行操作 |
| cf | 包装成DSCUDAFunction的kernel函数 |
| args | 需要传给GPU核函数的一系列自由变量 |
| outputArraySizes | 如果预期的返回结果是一个包装成线性形式的数组，为每个输出列指定数组长度的序列 |

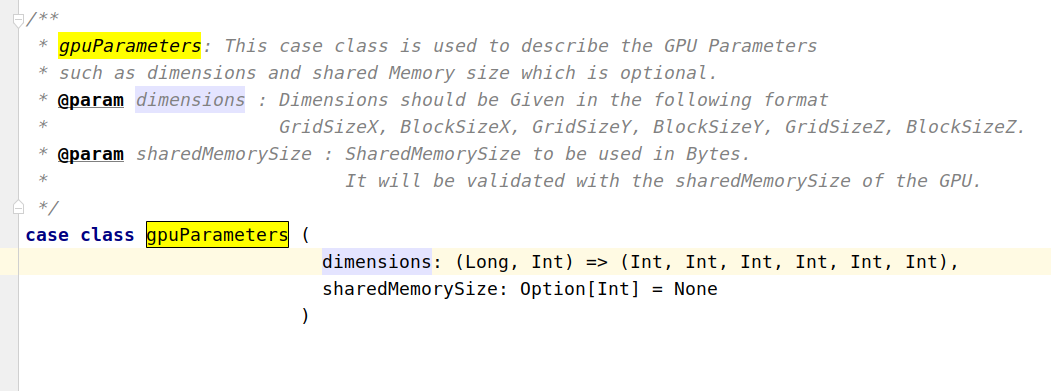
## 四．DSCUDAFunction



该case class用于描述CUDA核函数，并将核函数的输入、输出参数映射到应用此操作的DataSet的列名。可以指定stages和gpu核函数的dimensions。如果核函数要执行reduce的操作，则输出大小将与输入大小不同，因此用户必须提供outputSize。

参数说明：

|  |  |
| --- | --- |
| funcName | 核函数的函数名 |
| \_inputColumnsOrder | Dataset中，作为核函数输入参数的列名 |
| \_outputColumnsOrder | Dataset中，作为接收核函数返回结果的列名 |
| resource | 读取的Ptx文件，URL形式 |
| stagesCount |  |
| gpuParams | 核函数的维度，即blockSize和gridSize，还有sharedMemorySize |
| outputSize | 输出的大小 |

其中gpuParams的定义为

可以定义三维的grid和block和共享存储的大小。其中dimensions是一个lambda表达式，根据

（size:Long，stage:Int）=> (gridsize.x,blocksize.x,gridsize.y,blocksize.y,gridsize.z,blocksize.z)

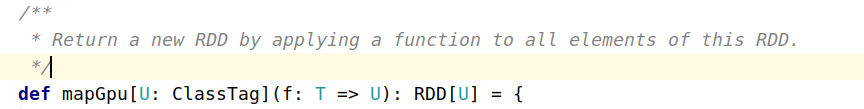
来决定kernel的维度。

# 自动生成gpu程序

使用mapGpu和reduceGpu两个函数，根据提供的lambda表达式，运行时生成ptx格式的gpu代码并执行，不需要自己写kernel函数，这个方法只适用于普通RDD。使用样例查看 GpuEnablerCodegen.scala

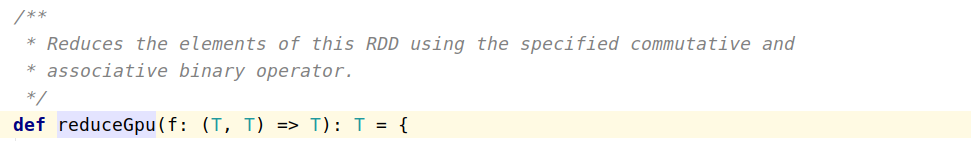
使用前要先设置 sparkConf.set("spark.gpu.codegen", "true")，否则无法生成gpu代码。

## mapGpu



对RDD中的每个元素进行f所代表的操作，其中f是lambda表达式的形式，gpu代码会在运行中生成并执行。

## reduceGpu



# Jar包使用方法

1.首先要把libJCudaDriver-10.1.0-linux-x86\_64.so

libJCudaRuntime-10.1.0-linux-x86\_64.so

libJNvrtc-10.1.0-linux-x86\_64.so

三个.so文件放到java.library.path中。Result文件夹下是编译好的结果。Gpuenabler文件夹是修改后的源码。

1. 终端进入result文件夹中，其中gpu-enabler\_2.11-2.0.0.jar是编译好的gpuenablerjar包，learn-gpuenabler.jar是把所有测例编译成的jar包。然后运行

spark-submit --jars gpu-enabler\_2.11-2.0.0.jar

--class com.ibm.gpuenabler.GpuKMeans

--master local[\*] learn-gpuenabler.jar

其中com.ibm.gpuenabler.GpuKMeans是要运行测例的名字，GpuKMeans可以换成其他程序。