kylin源码概要分析

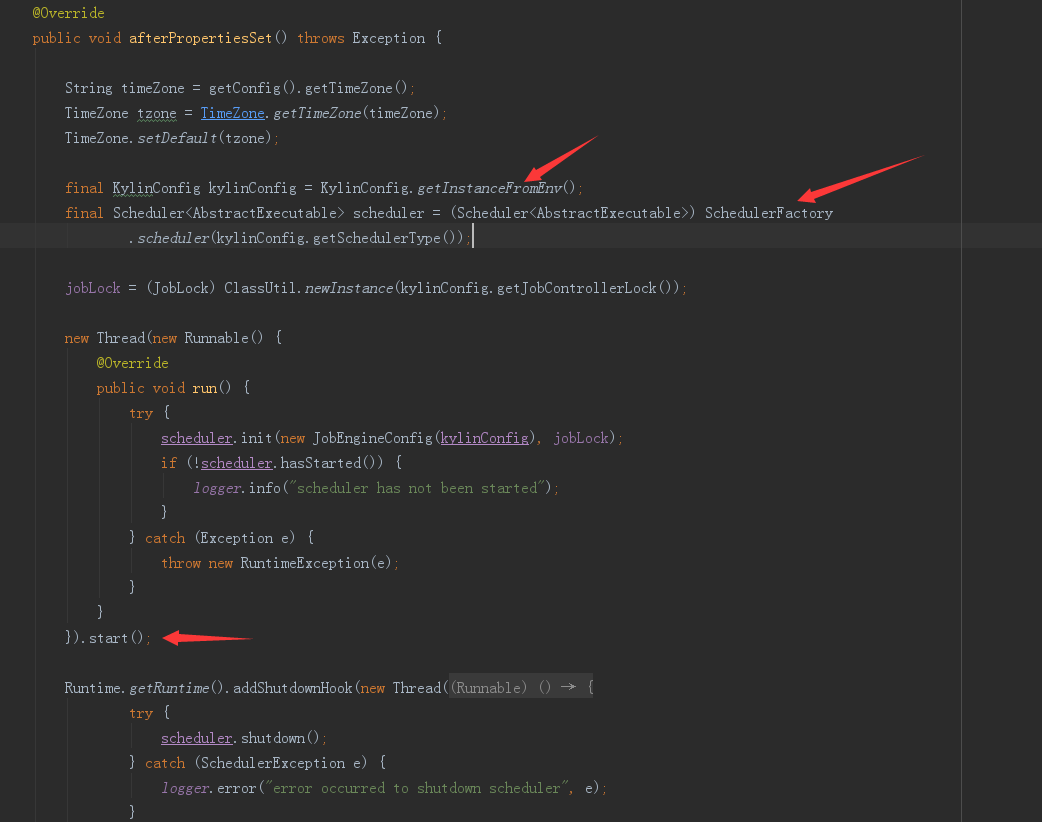
cube的构建分析

使用Kylin的时候，最重要的一步就是创建cube的模型定义，即指定度量和维度以及一些附加信息，然后对cube进行build，当然我们也可以根据原始表中的某一个string字段（这个字段的格式必须是日期格式，表示日期的含义）设定分区字段，这样一个cube就可以进行多次build，每一次的build会生成一个segment，每一个segment对应着一个时间区间的cube，这些segment的时间区间是连续并且不重合的，对于拥有多个segment的cube可以执行merge，相当于将一个时间区间内部的segment合并成一个。

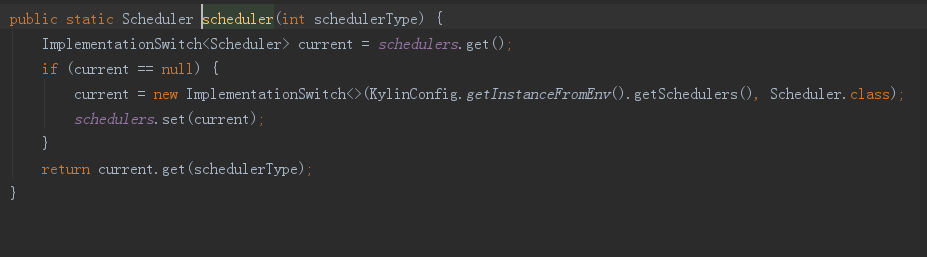
JobService定义

JobService实现了InitializingBean接口，这就意味着在初始化bean的时候spring会调用这个类实现的afterPropertiesSet方法执行初始化操作。在初始化函数中会加载kylin的配置文件（文件名为kylin.properties），配置文件的加载首先查找JVM的环境变量KYLIN\_CONF，如果不存在则进一步查找KYLIN\_HOME，找到其下面的conf目录，然后加载配置文件，并且还会查找该目录下的kylin.properties.override文件是否存在，如果存在则将后者的配置覆盖kylin.properties文件的配置。

创建调度



调度获取的方式采用配置文件中的schedulerType，通过工厂类获取。

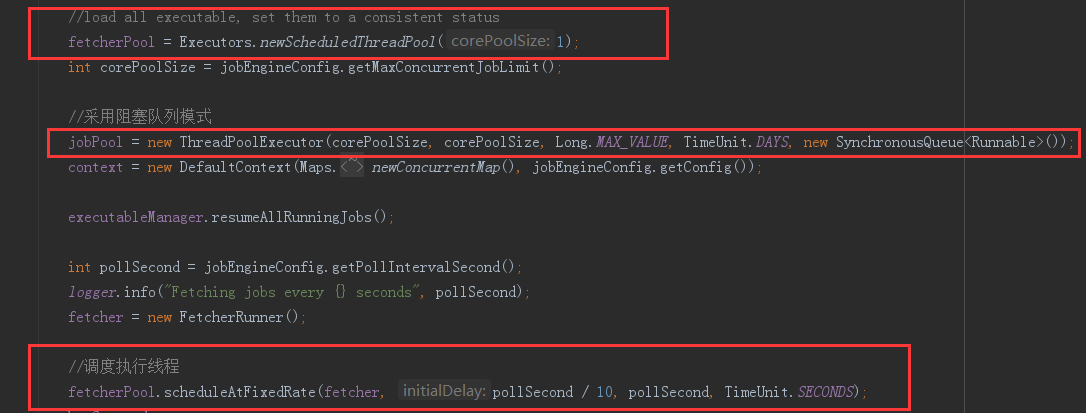


FetcherRunner和JobRunner

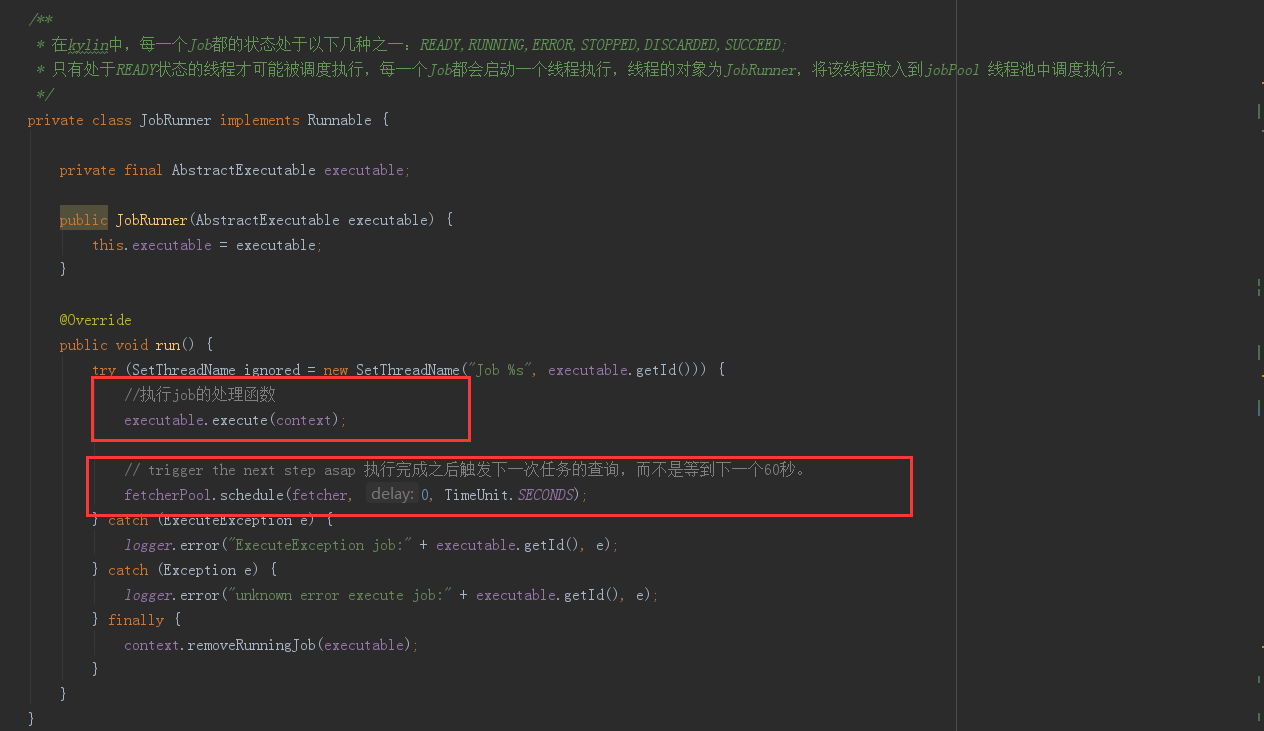
FetcherRunner是调度线程池，JobRunner是阻塞队列线程池，当Scheduler开始初始化时，FetcherRunner对象被创建，并开始调度执行，并执行了JobRunner，当JobRunner执行完成后，再一次触发调度线程，执行下一个JobRunner。

DefaultScheduler是默认的调度器，在init方法中，实现了2中线程池，并初始化。

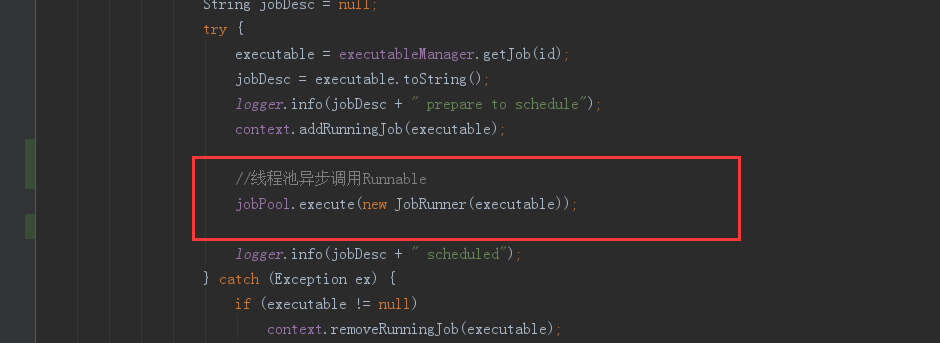




JobRunner具体的执行过程是



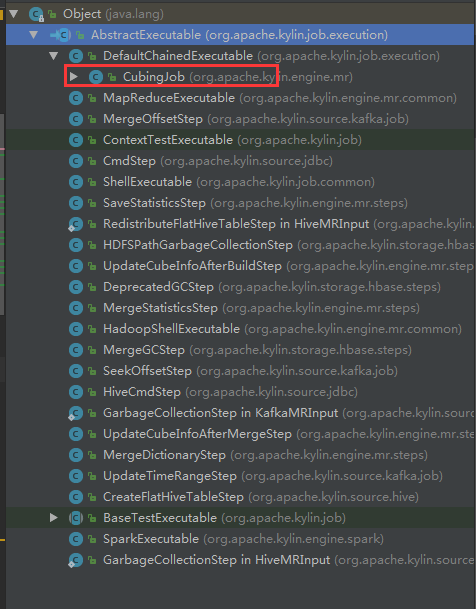
FetcherRunner提交JobRunner到线程池的流程如下



这种任务调度模型，启动了两个线程池(ScheduledExecutorService 和 fetcherPool ExecutorService jobPool)，线程池A只能持有一个线程，这个线程就是周期性的检查任务队列中是否有可执行的任务，如果存在可执行的任务则将它作为参数创建一个新的线程对象并交给线程池B调度执行，线程池B调度到这个线程的时候执行线程的run函数，该函数执行具体任务的逻辑，但是这些任务总体执行逻辑是相同的（先执行start，然后doWork最后执行finish），因此都继承自某一个虚拟类，在run函数中调用这个虚拟类的a函数，a函数内实现总体执行逻辑（start、doWork、finish等），然后由不同的子类实现不同的start、doWork、finish等具体函数已实现不同的逻辑。

Kylin Job介绍

     AbstractExecutable类的继承关系如下图



其中BaseTestExecutable我们不用关心，

DefaultChainedExecutable是用于链式执行多个job的job，它的内部可以链接多个job，然后按照job的加入顺序依次执行，它的子类包括CubingJob和IIJob分别用于实现构建普通的cube和构建倒排索引（这个暂时没有用到）。

     HadoopShellExecutable用于执行类似于在hadoop的shell下提交的mapreduce任务，它的参数必须包含一个mapreduce的job class，然后执行这个mapreduce任务，并且会同步等待这个任务的执行完成，只检查该任务是否执行成功。

     MapReduceExecutable同样也是为了执行一个mapreduce任务，但是和HadoopShellExecutable不同的是它使用异步的方式执行提交的mapreduce任务，它不会一直等待任务的执行完成，而是提交完成之后立即返回，然后周期性的访问hadoop任务状态的url（根据resourceManager地址）查看任务的执行状态，如果执行完成则会调用getCounter获取任务的一些统计信息，检查状态的时延可以由kylin.job.yarn.app.rest.check.interval.seconds配置项设置，默认为60s，所以在kylin中可以看到对于普通的任务一般使用HadoopShellExecutable执行，而对于关键的任务（cube构建过程中的任务）会使用MapReduceExecutable执行mapreduce任务。

     HqlExecutable用于执行多条hive sql语句，语句通过配置项传递，目前看到的唯一执行hive sql的地方在于创建cube的第一步需要从hive中获取原始数据，但是目前没有使用HqlExecutable，而是选择直接调用shell命令hive -e执行。

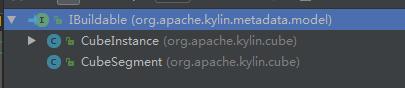
     ShellExecutable用于执行一个shell命令，如刚才提到的执行hive -e执行原始数据的生成。目前shell可以支持命令在当前主机执行或者在远端主机上执行，可以通过kylin.job.run.as.remote.cmd配置决定是否在远端执行，kylin.job.remote.cli.hostname指定了远端主机的主机名，kylin.job.remote.cli.username和kylin.job.remote.cli.password分别指定了登陆用户名和密码，实现是通过SSH登陆到远端主机再执行相应的命令。

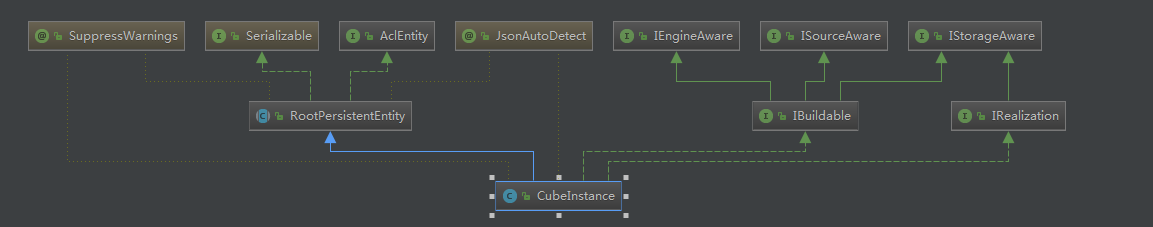
     UpdateCubeInfoAfterBuildStep和UpdateCubeInfoAfterMergeStep分别是用于在构建cube和合并cube完成之前的最后一步，这一步主要是更新一些统计信息，例如cube的大小，读写hdfs的大小等。

CubeInstance定义

CubeInstance定义一个cube对象，在create方法中主要设置了KylinConfig、cube的名称、cube的具体描述、cube创建的UTC时间、segments列表对象和cube的初始状态。

从CubeInstance的继承关系可以看出，cubeInstance和cubeSegment都实现了接口IBuildable。





CubeSegment定义

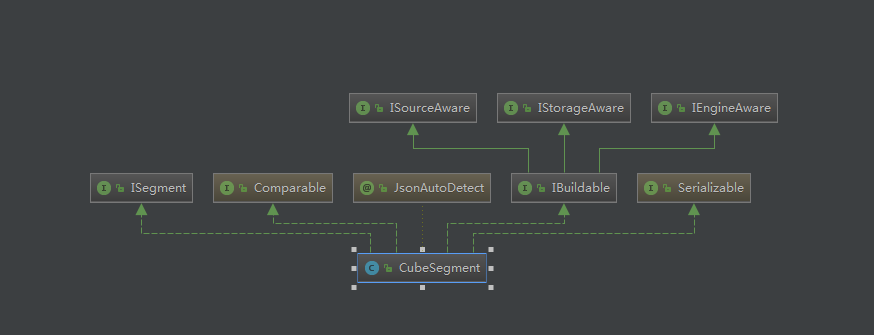
cube每一次的构建，都会生成一个segment，每一个segment对应着一个时间区间的cube，segment是可以进行merge操作的。

ISegment接口定义segment自身的属性

ISourceAware接口定义资源的来源， hive、streaming、sparkSQL、jdbc和external

IStorageAware接口定义segment存储，hbase、hybrid和shared hbase

IEngineAware接口定义计算引擎，mr1、mr2和spark

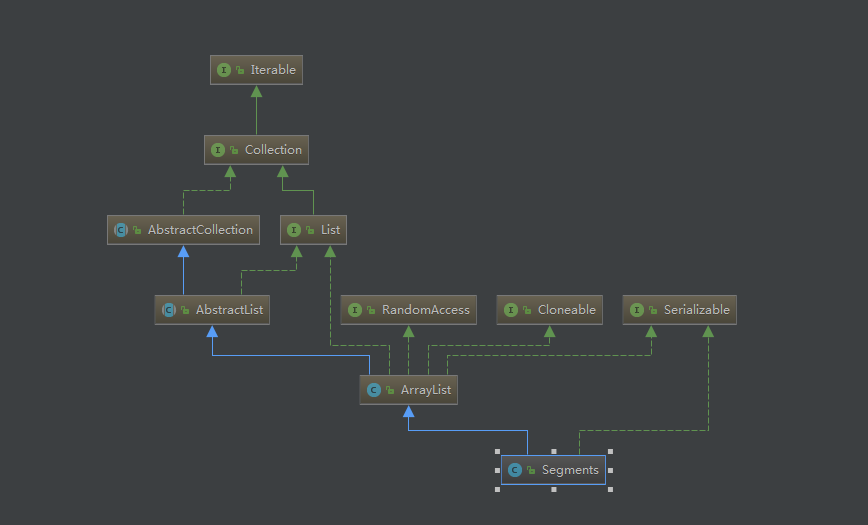


Segment类定义

segment继承ArrayList，主要是保存多个cube segment。

cube的build操作包括”BUILD”、”MERGE”和”REFRESH”，都是从segments中获取对应的cube segment进行下一步操作。

提供CubeSegment的列表保存，并提供多种获取cubeSegment的方式。



Cube构建流程

kylin的web页面上创建完成一个cube之后可以点击action下拉框执行build或者merge操作，这两个操作都会调用cube的rebuild接口，rest api提供的build也会调用rebuild接口。

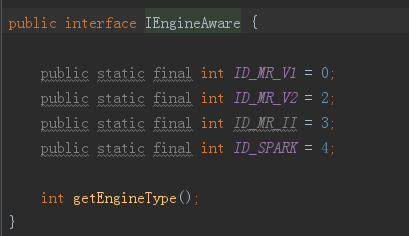
调用的参数包括：

* cube名，用于唯一标识一个cube，在当前的kylin版本中cube名是全局唯一的，而不是每一个project下唯一的；
* 本次构建的startTime和endTime，这两个时间区间标识本次构建的segment的数据源只选择这个时间范围内的数据；对于BUILD操作而言，startTime是不需要的，因为它总是会选择最后一个segment的结束时间作为当前segment的起始时间。
* buildType标识着操作的类型，可以是”BUILD”、”MERGE”和”REFRESH”。

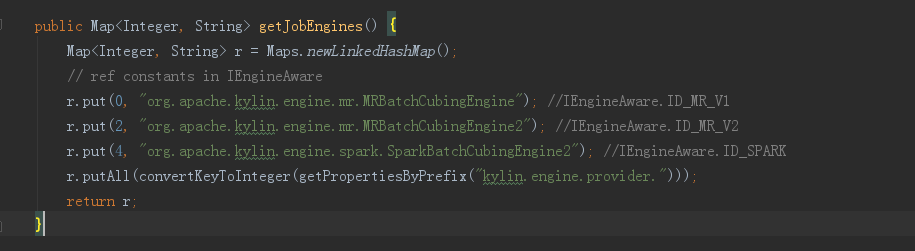
这些操作的统一入口就是JobService.submitJob函数。

JobService.submitJobInternal 接口首先获取资源的来源，再按照分区，创建一个新的segment。最后创建job引擎。

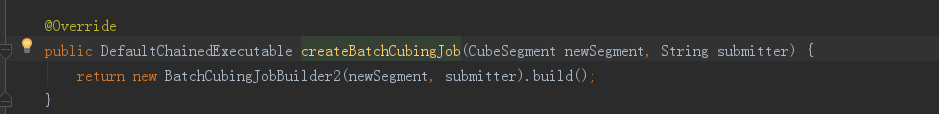
EngineFactory.createBatchCubingJob(newSeg, submitter); 根据segment的类继承关系发现，IEngineAware接口定义计算引擎，mr1、mr2和spark，根据EngineType通过EngineFactory来具体实现计算引擎。



通过工程模式创建对应的计算引擎，目前提供的计算引擎有

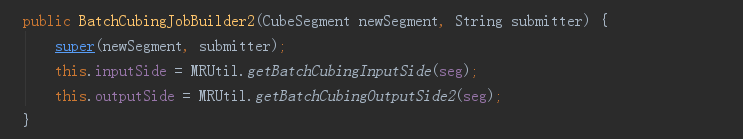


1.6版本采用的是MR2计算引擎。

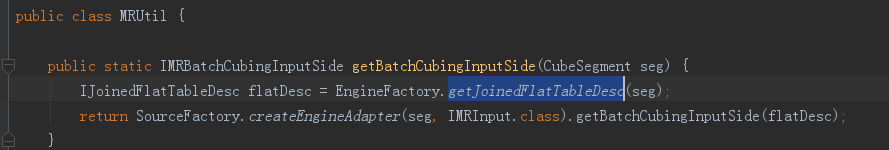


MRBatchCubingEngine.createBatchCubingJob 2接口非常简单，直接调用了build构建接口。

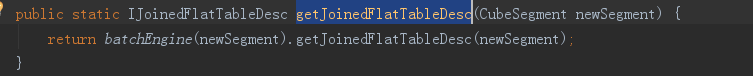
数据源、存储两个模块在BatchCubingJobBuilder2的初始化过程中。这两个属性就分别是数据源、存储在CubingJob构建中，与引擎进行协作。



获取过程：

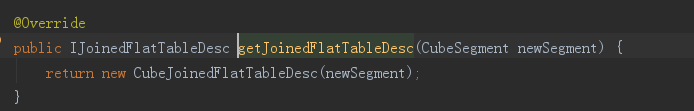


这个是获取inputSide的实例的过程，跟上面调用createBatchCubingJob方法的过程是一样的，最终调用的其实还是MRBatchCubingEngine2这个计算引擎的getJoinedFlatTableDesc方法。

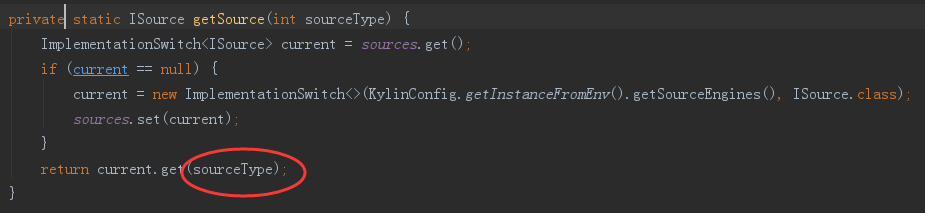


计算引擎MRBatchCubingEngine2的这个方法逻辑还是很简单，直接返回了一个CubeJoinedFlatTableDesc实例，

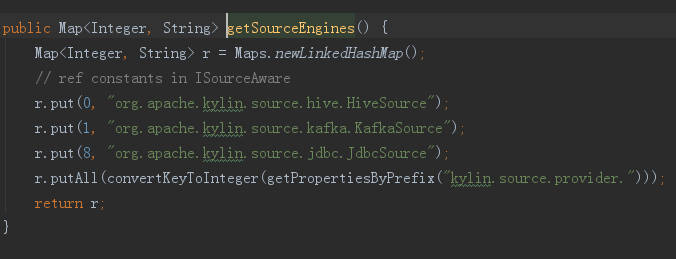
CubeJoinedFlatTableDesc对象可以理解成就是对数据源表的分装。

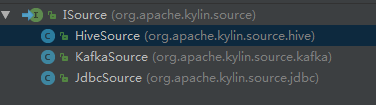
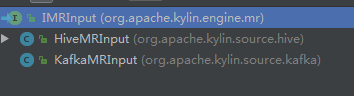


在获取flatDesc实例后，就要来获取inputSide的实例，这里和上面获取计算引擎代码非常相似，根据数据源类型通过工厂类实例化具体的类，如下：



目前Kylin中支持的数据源有, 在Kylin中，离线数据源，采用HiveSource，所以接下来的分析中，也都是以HiveSource为例。



SourceFactory的createEngineAdapter，实际调用的是HiveSource的adaptToBuildEngine，根据传入的IMRInput.class接口，最终返回的是HiveMRInput的实例。

流程图

创建CubingJob对象，CubingJob继承DefaultChainedExecutable，实现链式job调用。CubingJob不参与具体job的执行，只处理其他job的调度。

kylin重新生成Hive表，采用SequenceFile压缩表数据：step1 创建平面表。

创建Hive外部表，并采用SequenceFile格式压缩文件

