20180327_NameNode, DataNode 和 Job tracker， task tracker

1- NMAENODE有两种容错机制：
    1- 数据本地化
    2- 主备节点(通过网络把 元数据拷贝到备用节点)

2- 备用节点会做一些日志的合并工作，防止日志文件过大。
   通常 SecondaryNameNode 运行在一个单独的物理机上

3- 后台程序NameNode、Secondary NameNode、JobTracker运行在Master节点上，而在每个Slave节点上，部署一个DataNode和TaskTracker，以便 这个Slave服务器运行的数据处理程序能尽可能直接处理本机的数据。对Master节点需要特别说明的是，在小集群中，Secondary NameNode可以属于某个从节点；在大型集群中，NameNode和JobTracker被分别部署在两台服务器上**。

4- 五个进程：

5- DataNode：
    根据客户端或者是namenode的调度存储和检索数据，并且定期向namenode发送他们所存储的块(block)的列表。集群中的每个服务器都运行一个DataNode后台程序，这个后台程序负责把HDFS数据块读写到本地的文件系统

6- Second Name Node：
    Secondary NameNode是一个用来监控HDFS状态的辅助后台程序。就想NameNode一样，每个集群都有一个Secondary NameNode，并且部署在一个单独的服务器上。Secondary NameNode不同于NameNode，它不接受或者记录任何实时的数据变化，但是，它会与NameNode进行通信，以便定期地保存HDFS元数据的 快照。由于NameNode是单点的，通过Secondary NameNode的快照功能，可以将NameNode的宕机时间和数据损失降低到最小.

7- 客户端作业的提交：
    JobClient除了自己完成一部分必要的工作外【jobClient 提交作业】，还负责与JobTracker进行交互 JobClient在获取了JobTracker为Job分配的id之后，会在JobTracker的系统目录(HDFS)下为该Job创建一个单独的目录，目录的名字即是Job的id，该目录下会包含文件job.xml、job.jar、job.split等，其中，job.xml文件记录了Job的详细配置信息，job.jar保存了用户定义的关于job的map、reduce操纵，job.split保存了job任务的切分信息。在上面的流程图中，我想详细阐述的是JobClient是任何配置Job的运行环境，以及如何对Job的输入数据进行切分。
8- 整个的过程
    输入--> split(分片) --> map --> combine,shuffle ---> reduce ---> partition -----> 输出

```
9- Job tracker 的工作
    一.为作业生成一个Job；
    二.接受该作业。
    客户端的JobClient向JobTracker正式提交作业时直传给了它一个改作业的JobId，这是因为与Job相关的所有信息已经存在于JobTracker的系统目录下，JobTracker只要根据JobId就能得到这个Job目录。


10- Task Tracker 三大组件
    服务组件：http Sever 和 进度报告
    管理组件：负责对该节点上的任务、作业、JVM实例以及内存进行管理，
    工作组件：则负责调度Map/Reduce任务的执行
```

问题导读： 1.job的本质是什么?

2.任务的本质是什么?

3.文件系统的Namespace由谁来管理，Namespace的作用是什么?

4.Namespace 镜像文件(Namespace image)和操作日志文件(edit log)文件的作用是什么?

5.Namenode记录着每个文件中各个块所在的数据节点的位置信息，但是他并不持久化存储这些信息，为什么?

6.客户端读写某个数据时，是否通过NameNode?

7.namenode，datanode，Namespace image，Edit log之间的关系是什么?

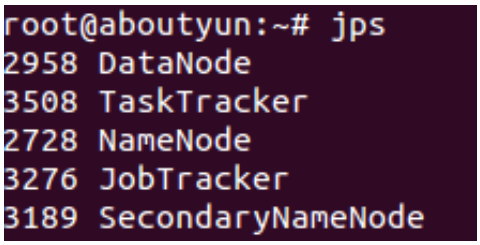
8.一旦某个task失败了，JobTracker如何处理?

9.JobClient JobClient在获取了JobTracker为Job分配的id之后，会在JobTracker的系统目录(HDFS)下为该Job创建一个单独的目录，目录的名字即是Job的id，该目录下会包含文件job.xml、job.jar等文件，这两个文件的作用是什么?

10.JobTracker根据什么就能得到这个Job目录?

11.JobTracker提交作业之前，为什么要检查内存? 12.每个TaskTracker产生多个java 虚拟机（JVM）的原因是什么?

概述：



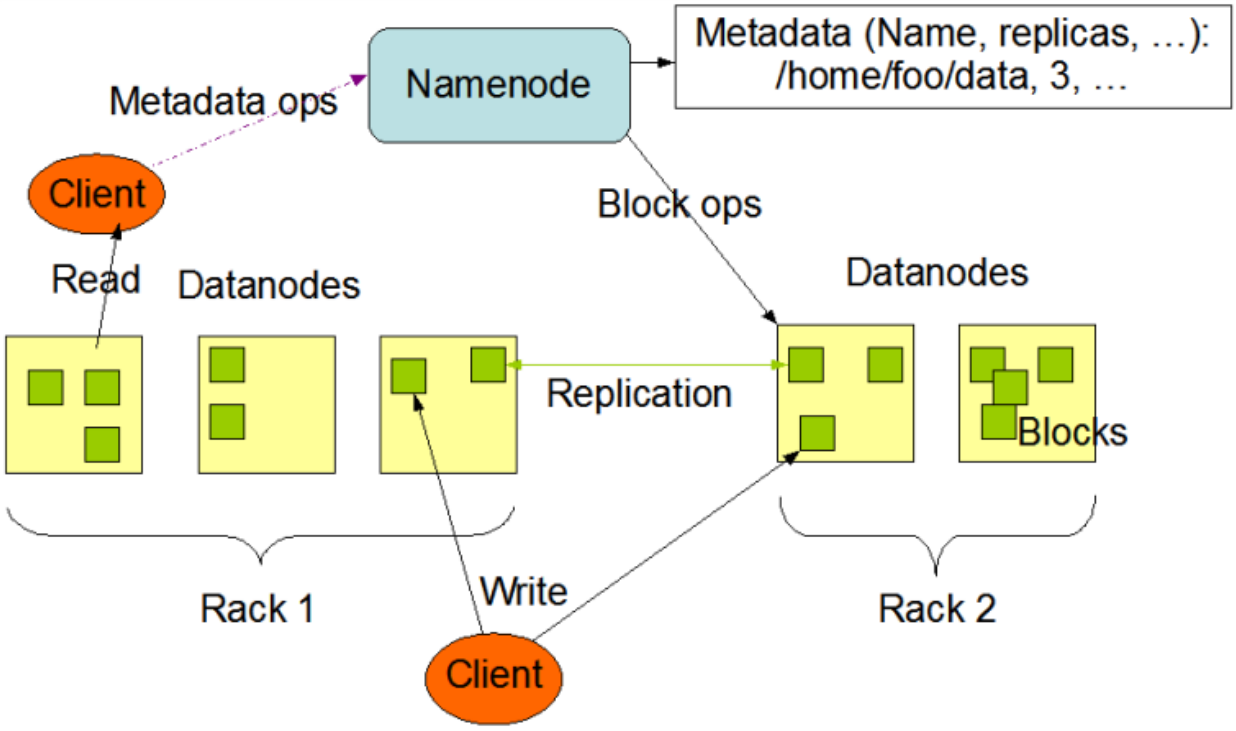
Hadoop是一个能够对大量数据进行分布式处理的软件框架，实现了Google的MapReduce编程模型和框架，能够把应用程序分割成许多的 小的工作单元，并把这些单元放到任何集群节点上执行。在MapReduce中，一个准备提交执行的应用程序称为“作业（job）”，而从一个作业划分出 得、运行于各个计算节点的工作单元称为“任务（task）”。此外，Hadoop提供的分布式文件系统（HDFS）主要负责各个节点的数据存储，并实现了 高吞吐率的数据读写。

在分布式存储和分布式计算方面，Hadoop都是用从/从（Master/Slave）架构。在一个配置完整的集群上，想让Hadoop这头大 象奔跑起来，需要在集群中运行一系列后台(deamon）程序。不同的后台程序扮演不用的角色，这些角色由NameNode、DataNode、 Secondary NameNode、JobTracker、TaskTracker组成。其中**NameNode、Secondary NameNode、JobTracker运行在Master节点上，而在每个Slave节点上，部署一个DataNode和TaskTracker，以便 这个Slave服务器运行的数据处理程序能尽可能直接处理本机的数据**。对Master节点需要特别说明的是，**在小集群中，Secondary NameNode可以属于某个从节点；在大型集群中，**NameNode和JobTracker被分别部署在两台服务器上**。

我们已经很熟悉这个**5个进程**，但是在使用的过程中，我们经常遇到问题，那么该如何入手解决这些问题。那么首先我们需了解的他们的原理和作用。

1.Namenode介绍

Namenode 管理者文件系统的Namespace。它维护着文件系统树(filesystem tree)以及文件树中所有的文件和文件夹的元数据(metadata)。管理这些信息的文件有两个，分别是Namespace 镜像文件(Namespace image)和操作日志文件(edit log)，这些信息被Cache在RAM中，当然，这两个文件也会被持久化存储在本地硬盘。Namenode记录着每个文件中各个块所在的数据节点的位置信息，但是他并不持久化存储这些信息，因为这些信息会在系统启动时从数据节点重建。 Namenode结构图课抽象为如图：



客户端(client)代表用户与namenode和datanode交互来访问整个文件系统。客户端提供了一些列的文件系统接口，因此我们在编程时，几乎无须知道datanode和namenode，即可完成我们所需要的功能。

1.1Namenode容错机制

没有Namenode，HDFS就不能工作。事实上，如果运行namenode的机器坏掉的话，系统中的文件将会完全丢失，因为没有其他方法能够将位于不同datanode上的文件块(blocks)重建文件。因此，namenode的容错机制非**常重要，Hadoop提供了两种机制。 第一种方式是将持久化存储在本地硬盘的文件系统元数据备份。Hadoop可以通过配置来让Namenode将他的持久化状态文件写到不同的文件系统中。这种写操作是同步并且是原子化的。比较常见的配置是在将持久化状态写到本地硬盘的同时，也写入到一个远程挂载的网络文件系统。 第二种方式是运行一个辅助的Namenode(Secondary Namenode)。 事实上Secondary Namenode并不能被用作Namenode它的主要作用是定期的将Namespace镜像与操作日志文件(edit log)合并，以防止操作日志文件(edit log)变得过大。通常，Secondary Namenode 运行在一个单独的物理机上，因为合并操作需要占用大量的CPU时间以及和Namenode相当的内存。辅助Namenode保存着合并后的Namespace镜像的一个备份，万一哪天Namenode宕机了，这个备份就可以用上了。** 但是辅助Namenode总是落后于主Namenode，所以在Namenode宕机时，数据丢失是不可避免的。在这种情况下，一般的，要结合第一种方式中提到的远程挂载的网络文件系统(NFS)中的Namenode的元数据文件来使用，把NFS中的Namenode元数据文件，拷贝到辅助Namenode，并把辅助Namenode作为主Namenode来运行。

2、Datanode介绍

Datanode是文件系统的工作节点，他们根据客户端或者是namenode的调度存储和检索数据，并且定期向namenode发送他们所存储的块(block)的列表。集群中的每个服务器都运行一个DataNode后台程序，这个后台程序负责把HDFS数据块读写到本地的文件系统。当需要通过客户端读/写某个 数据时，先由NameNode告诉客户端去哪个DataNode进行具体的读/写操作，然后，客户端直接与这个DataNode服务器上的后台程序进行通 信，并且对相关的数据块进行读/写操作。

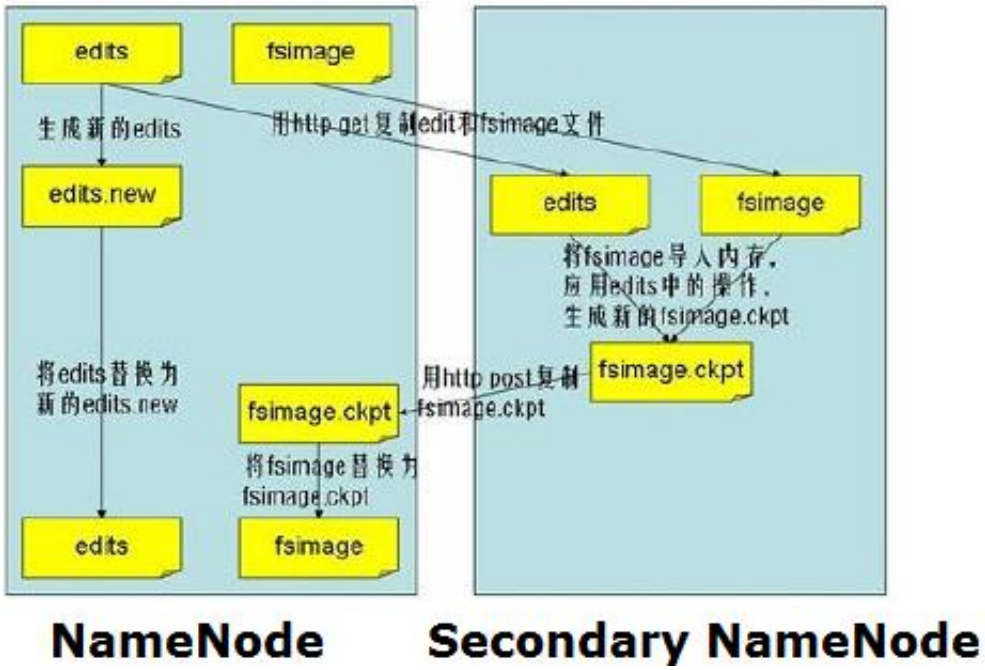
3、Secondary NameNode介绍

Secondary NameNode是一个用来**监控H**DFS状态的辅助后台程序。就想NameNode一样，每个集群都有一个Secondary NameNode，并且部署在一个单独的服务器上。Secondary NameNode不同于NameNode，它不接受或者记录任何实时的数据变化，但是，它会与NameNode进行通信，以便定期地保存HDFS元数据的 快照。由于NameNode是单点的，通过Secondary NameNode的快照功能，可以将NameNode的宕机时间和数据损失降低到最小。同时，如果NameNode发生问题，Secondary NameNode可以及时地作为备用NameNode使用。

3.1NameNode的目录结构如下：

${dfs.name.dir}/current/VERSION /edits /fsimage /fstime 3.2Secondary NameNode的目录结构如下：

${fs.checkpoint.dir}/current/VERSION /edits /fsimage /fstime /previous.checkpoint/VERSION /edits /fsimage /fstime



如上图，Secondary NameNode主要是做Namespace image和Edit log合并的。 那么这两种文件是做什么的？当客户端执行写操作，则NameNode会在edit log记录下来，（我感觉这个文件有些像Oracle的online redo logo file）并在内存中保存一份文件系统的元数据。 Namespace image（fsimage）文件是文件系统元数据的持久化检查点，不会在写操作后马上更新，因为fsimage写非常慢（这个有比较像datafile）。

由于Edit log不断增长，在NameNode重启时，会造成长时间NameNode处于安全模式，不可用状态，是非常不符合Hadoop的设计初衷。所以要周期性合并Edit log，但是这个工作由NameNode来完成，会占用大量资源，这样就出现了Secondary NameNode，它可以进行image检查点的处理工作。步骤如下：

（1） Secondary NameNode请求NameNode进行edit log的滚动（即创建一个新的edit log），将新的编辑操作记录到新生成的edit log文件；

（2） 通过http get方式，读取NameNode上的fsimage和edits文件，到Secondary NameNode上；

（3） 读取fsimage到内存中，即加载fsimage到内存，然后执行edits中所有操作（类似OracleDG，应用redo log），并生成一个新的fsimage文件，即这个检查点被创建；

（4） 通过http post方式，将新的fsimage文件传送到NameNode；

（5） NameNode使用新的fsimage替换原来的fsimage文件，让（1）创建的edits替代原来的edits文件；并且更新fsimage文件的检查点时间。整个处理过程完成。Secondary NameNode的处理，是将fsimage和edites文件周期的合并，不会造成nameNode重启时造成长时间不可访问的情况。

4、JobTracker介绍

JobTracker后台程序用来连接应用程序与Hadoop。用户代码提交到集群以后，**由JobTracker决定哪个文件将被处理，并且为 不同的task分配节点。同时，它还监控所有的task，一旦某个task失败了，JobTracker就会自动重新开启这个task，在大多数情况下这 个task会被放在不用的节点上**。每个Hadoop集群只有一个JobTracker，一般运行在集群的Master节点上。 下面我们详细介绍：
4.1JobClient

我们配置好作业之后，就可以向JobTracker提交该作业了，然后JobTracker才能安排适当的TaskTracker来完成该作业。那么MapReduce在这个过程中到底做了那些事情呢？这就是本文以及接下来的一片博文将要讨论的问题，当然本文主要是**围绕客户端在作业的提交过程中的工作**来展开。先从全局来把握这个过程吧！

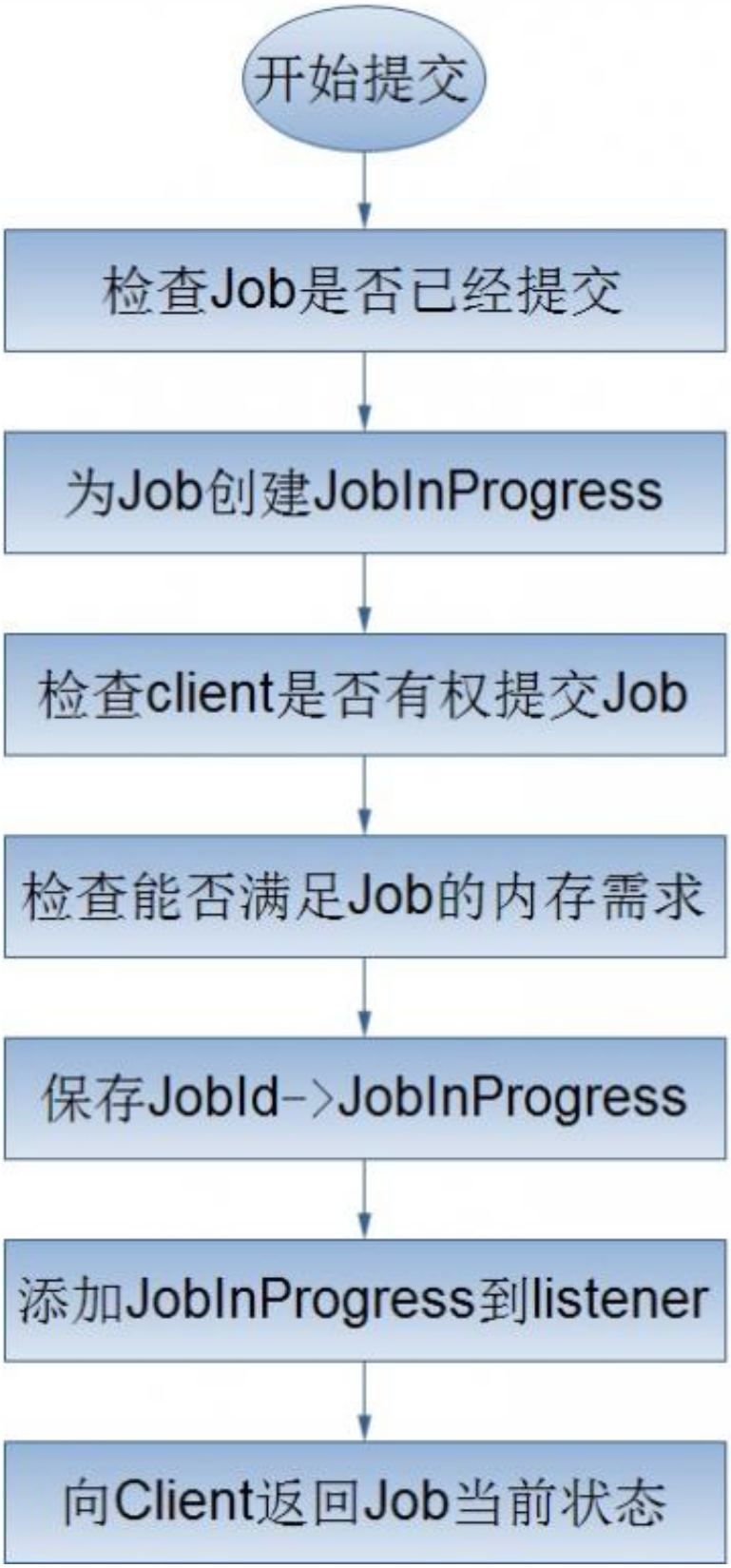
在Hadoop中，作业是使用Job对象来抽象的，对于Job，我首先不得不介绍它的一个大家伙**JobClient——客户端的实际工作者。JobClient除了自己完成一部分必要的工作外，还负责与JobTracker进行交互**。所以客户端对Job的提交，绝大部分都是JobClient完成的，从上图中，我们可以得知JobClient提交Job的详细流程主要如下：

JobClient在获取了JobTracker为Job分配的id之后，会在JobTracker的系统目录(HDFS)下为该Job创建一个单独的目录，目录的名字即是Job的id，该目录下会包含文件job.xml、job.jar、job.split等，其中，job.xml文件记录了Job的详细配置信息，job.jar保存了用户定义的关于job的map、reduce操纵，job.split保存了job任务的切分信息。在上面的流程图中，我想详细阐述的是JobClient是任何配置Job的运行环境，以及如何对Job的输入数据进行切分。

4.2JobTracker

上面谈到了客户端的JobClient对一个作业的提交所做的工作，那么这里，就要好好的谈一谈JobTracker为作业的提交到底干了那些个事情——**一.为作业生成一个Job；二.接受该作业。**

我们都知道，客户端的JobClient把作业的所有相关信息都保存到了JobTracker的系统目录下(当然是HDFS了)，这样做的一个最大的好处就是客户端干了它所能干的事情同时也减少了服务器端JobTracker的负载。下面就来看看JobTracker是如何来完成客户端作业的提交的吧！哦。对了，在这里我不得不提的是客户端的JobClient向JobTracker正式提交作业时直传给了它一个改作业的JobId，这是因为与Job相关的所有信息已经存在于JobTracker的系统目录下，JobTracker只要根据JobId就能得到这个Job目录。



对于上面的Job的提交处理流程，我将简单的介绍以下几个过程：

1.创建Job的JobInProgress

JobInProgress对象详细的记录了Job的配置信息，以及它的执行情况，确切的来说应该是Job被分解的map、reduce任务。在JobInProgress对象的创建过程中，它主要干了两件事，一是把Job的job.xml、job.jar文件从Job目录copy到JobTracker的本地文件系统(job.xml->*/jobTracker/jobid.xml*, *job.jar*->/jobTracker/jobid.jar)；二是创建JobStatus和Job的mapTask、reduceTask存队列来跟踪Job的状态信息。

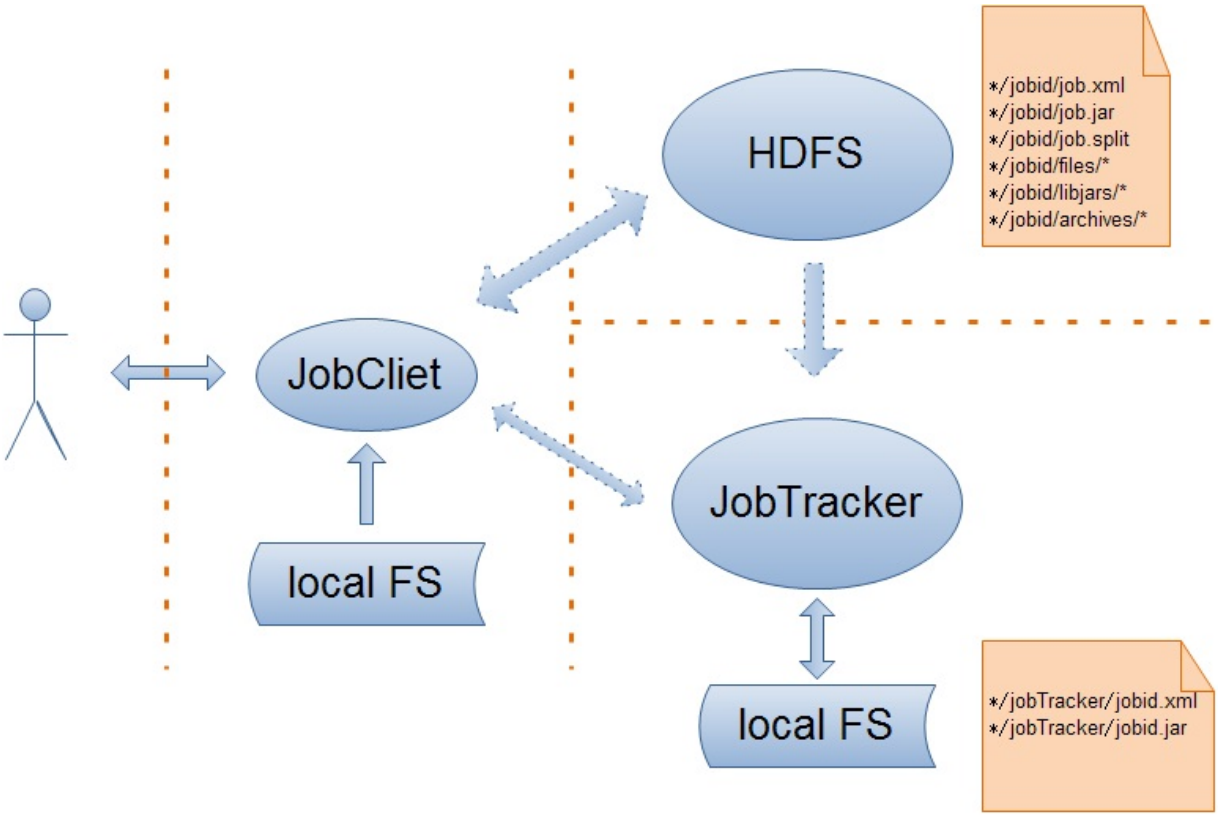
2.检查客户端是否有权限提交Job

JobTracker验证客户端是否有权限提交Job实际上是交给QueueManager来处理的。

3.检查当前mapreduce集群能够满足Job的内存需求

客户端提交作业之前，会根据实际的应用情况配置作业任务的内存需求，同时JobTracker为了提高作业的吞吐量会限制作业任务的内存需求，所以在Job的提交时，JobTracker需要**检查Job的内存需求是否满足JobTracker的设置。**

上面流程已经完毕，可以总结为下图：



5、TaskTracker介绍

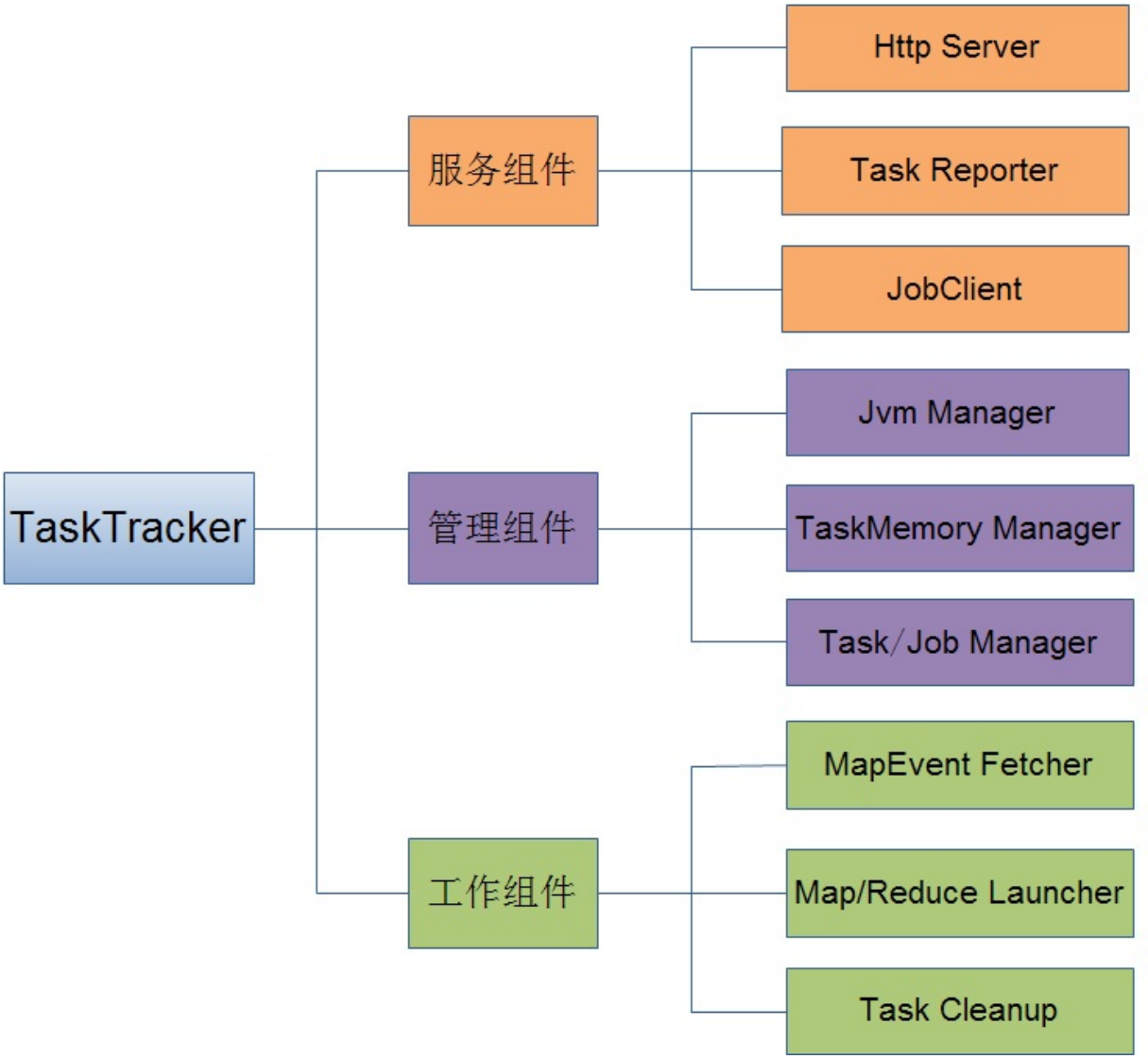
TaskTracker与负责存储数据的DataNode相结合，其处理结构上也遵循主/从架构。**JobTracker位于主节点，统领 MapReduce工作；而TaskTrackers位于从节点，独立管理各自的task。**每个TaskTracker负责独立执行具体的task，而 JobTracker负责分配task。虽然每个从节点仅有一个唯一的一个TaskTracker，但是每个TaskTracker可以产生多个java 虚拟机（JVM），用于并行处理多个map以及reduce任务。TaskTracker的一个重要职责就是与JobTracker交互。如果 JobTracker无法准时地获取TaskTracker提交的信息，JobTracker就判定TaskTracker已经崩溃，并将任务分配给其他 节点处

理。

5.1TaskTracker内部设计与实现

Hadoop**采用master-slave的架构设计来实现Map-Reduce框架**，它的JobTracker节点作为主控节点来管理和调度用户提交的作业，TaskTracker节点作为工作节点来负责执行JobTracker节点分配的Map/Reduce任务。**整个集群由一个JobTracker节点和若干个TaskTracker节点组成**，当然，JobTracker节点也负责对TaskTracker节点进行管理。在前面一系列的博文中，我已经比较系统地讲述了JobTracker节点内部的设计与实现，而在本文，我将对TaskTracker节点的内部设计与实现进行一次全面的概述。

TaskTracker节点作为工作节点不仅要和JobTracker节点进行频繁的交互来获取作业的任务并负责在本地执行他们，而且也要和其它的TaskTracker节点交互来协同完成同一个作业。因此，在目前的Hadoop-0.20.2.0实现版本中，对工作节点TaskTracker的设计主要包含三类组件：**服务组件、管理组件、工作组件**。服务组件不仅负责与其它的TaskTracker节点而且还负责与JobTracker节点之间的通信服务，**管理组件负责对该节点上的任务、作业、JVM实例以及内存进行管理，工作组件则负责调度Map/Reduce任务的执行**。这三大组件的详细构成如下：



下面来详细的介绍这三类组件：

服务组件

TaskTracker节点内部的服务组件不仅用来为TaskTracker节点、客户端提供服务，而且还负责向TaskTracker节点请求服务，这一类组件主要包括HttpServer、TaskReportServer、JobClient三大组件。

1.HttpServer

TaskTracker节点在其内部使用Jetty Web容器来开启http服务，这个http服务一是用来为客户端提供Task日志查询服务，二是用来提供数据传输服务，即在执行Reduce任务时是通过TaskTracker节点提供的该http服务来获取属于自己的map输出数据。这里需要详细介绍的是与该服务相关的配置参数，集群管理者可以通过TaskTracker节点的配置文件来配置该服务地址和端口号，对应的配置项为：mapred.task.tracker.http.address。同时，为了能够灵活的控制该该服务的吞吐量，管理者还可以设置该http服务的内部工作线程数量，对应的配置为：tasktracker.http.threads。

2.Task Reporter

TaskTracker节点在接收到JobTracker节点发送过来的Map/Reduce任务之后，会把它们交给JVM实例来执行，而自己则需要收集这些任务的执行进度信息，这就使得Task在JVM实例中执行的时候需要不断地向TaskTracker节点报告当前的执行情况。虽然TaskTracker节点和JVM实例在同一台机器上，但是它们之间的进程通信却是通过网络I/O来完成的(此处并不讨论这种通信方式的性能)，也就是TaskTracker节点在其内部开启一个端口来专门为任务实例提供进度报告服务。该服务地址可以通过配置项mapred.task.tracker.report.address来设置，而服务内部的工作线程的数量取2倍于该TaskTracker节点上的Map/Reduce Slot数量中的大者。