> 多为几个为什么

Spark通信架构和集群启动流程

Spark内核概述

> Spark内核泛指Spark的核心运行机制，包括Spark核心组件的运行机制、Spark任务调度机制、Spark内存管理机制、Spark核心功能的运行原理等，熟练掌握Spark内核原理，能够帮助我们更好地完成Spark代码设计，并能够帮助我们准确锁定项目运行过程中出现的问题的症结所在。

Spark核心组件

Driver

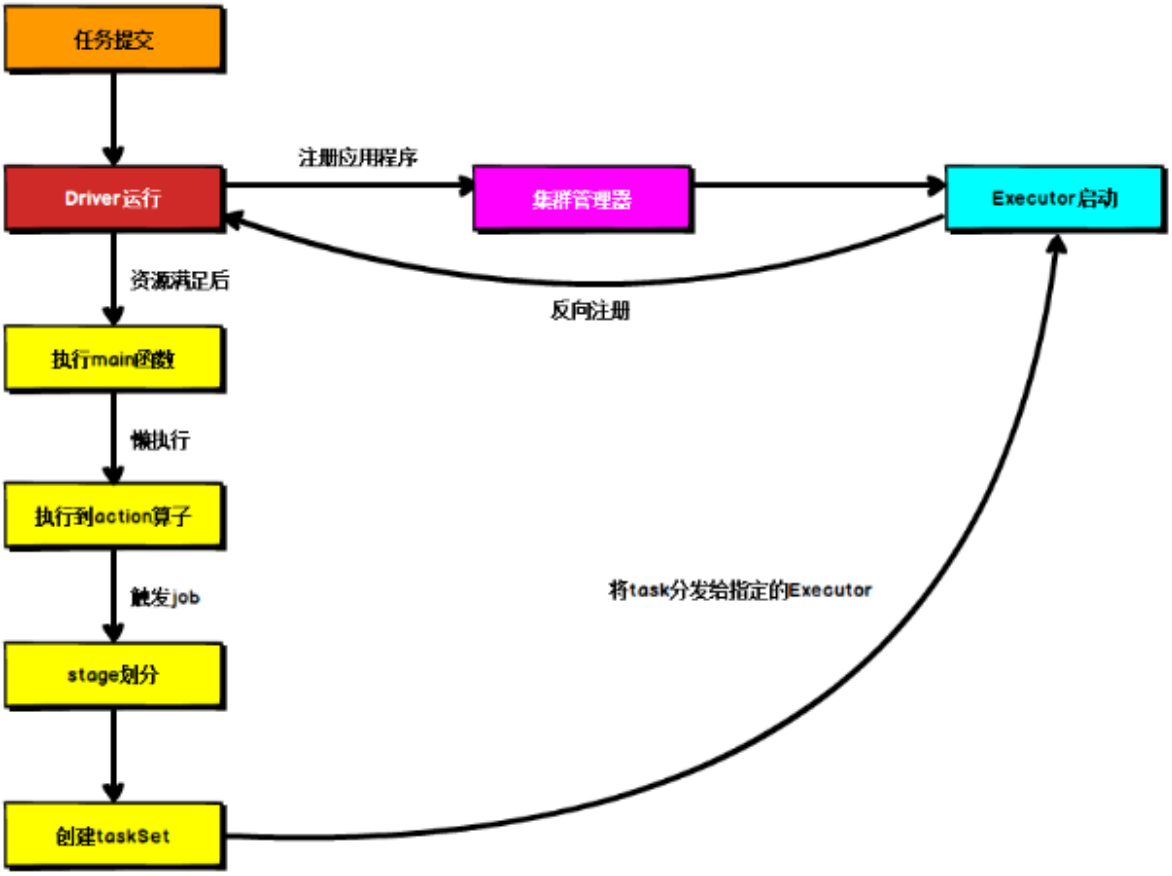
Spark驱动器节点，用于执行Spark任务中的main方法，负责实际代码的执行工作。Driver在Spark作业执行时主要负责：
（1）将用户程序转化为作业（Job）；
（2）在Executor之间调度任务（Task）；
（3）跟踪Executor的执行情况；
（4）通过UI展示查询运行情况；

Executor

Spark Executor节点是负责在 Spark 作业中运行具体任务，任务彼此之间相互独立。Spark 应用启动时，Executor节点被同时启动，并且始终伴随着整个 Spark 应用的生命周期而存在。如果有Executor节点发生了故障或崩溃，Spark 应用也可以继续执行，会将出错节点上的任务调度到其他Executor节点上继续运行。

Executor有两个核心功能：
（1）负责运行组成Spark应用的任务，并将结果返回给驱动器（Driver）；
（2）它们通过自身的块管理器（Block Manager）为用户程序中要求缓存的 RDD 提供内存式存储。RDD是直接缓存在Executor进程内的，因此任务可以在运行时充分利用缓存数据加速运算。

Spark通用运行流程概述



> 上图为Spark通用运行流程，不论Spark以何种模式进行部署，都是以如下核心步骤进行工作的：

（1）任务提交后，都会先启动Driver程序；

（2）随后Driver向集群管理器注册应用程序；

（3）之后集群管理器根据此任务的配置文件分配Executor并启动；

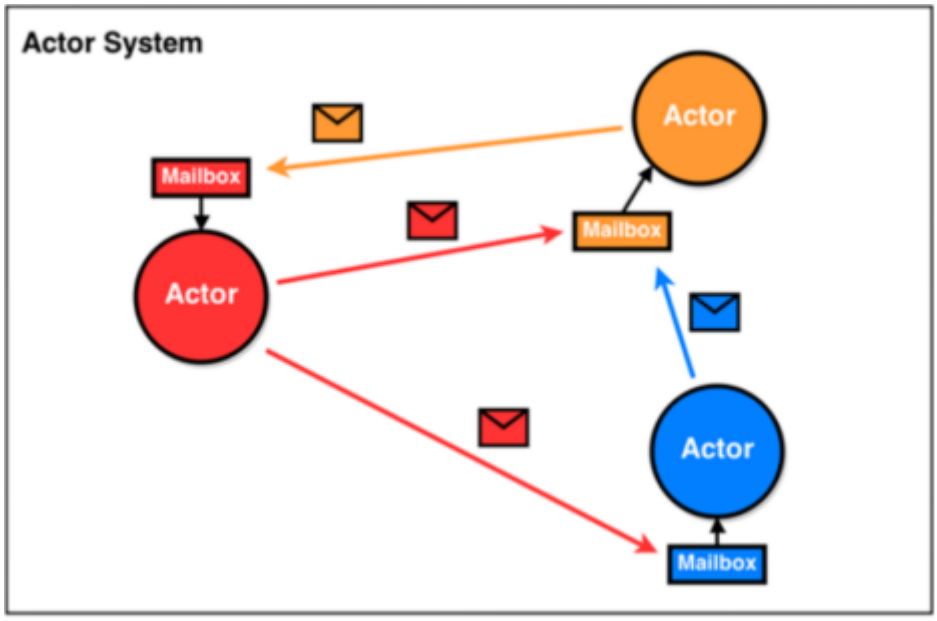
（4）当Driver所需的资源全部满足后，Driver开始执行main函数，Spark查询为懒执行，当执行到Action算子时开始反向推算，根据宽依赖进行Stage的划分，随后每一个Stage对应一个Taskset，Taskset中有多个Task；

（5）根据本地化原则，Task会被分发到指定的Executor去执行，在任务执行的过程中，Executor也会不断与Driver进行通信，报告任务运行情况。

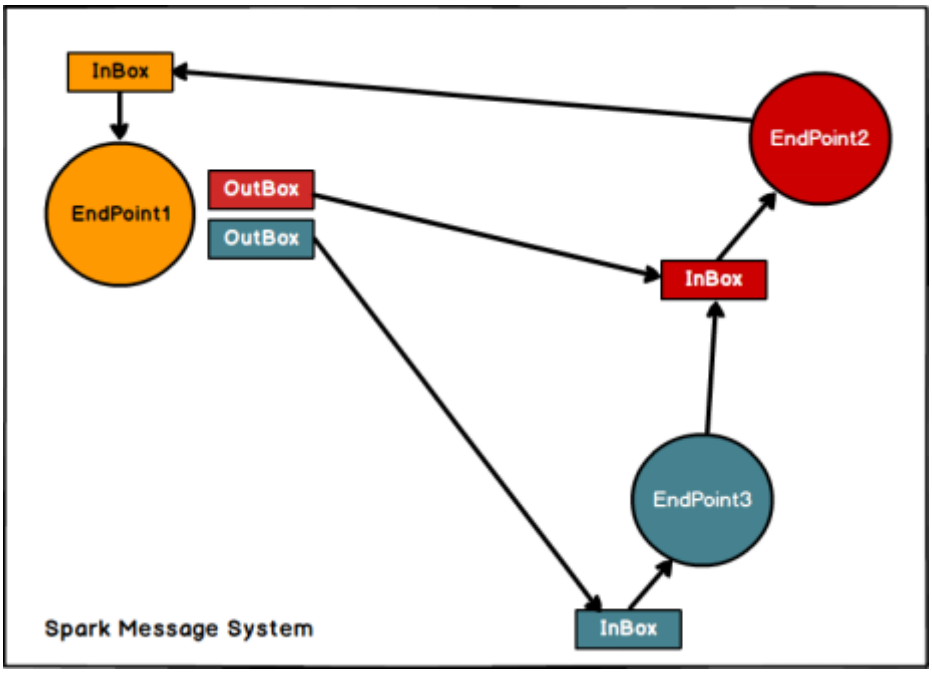
Spark通讯架构

通讯架构发展

> Spark中通信框架的发展：
> （1）Spark早期版本中采用Akka作为内部通信部件。
> （2）Spark1.3中引入Netty通信框架，为了解决Shuffle的大数据传输问题使用
> （3）Spark1.6中Akka和Netty可以配置使用。Netty完全实现了Akka在Spark中的功能。
> （4）Spark2系列中，Spark抛弃Akka，使用Netty。
> Spark2.x版本使用Netty通讯框架作为内部通讯组件。Spark 基于Netty新的RPC框架借鉴了Akka的中的设计，它是基于Actor模型，如下图所示：



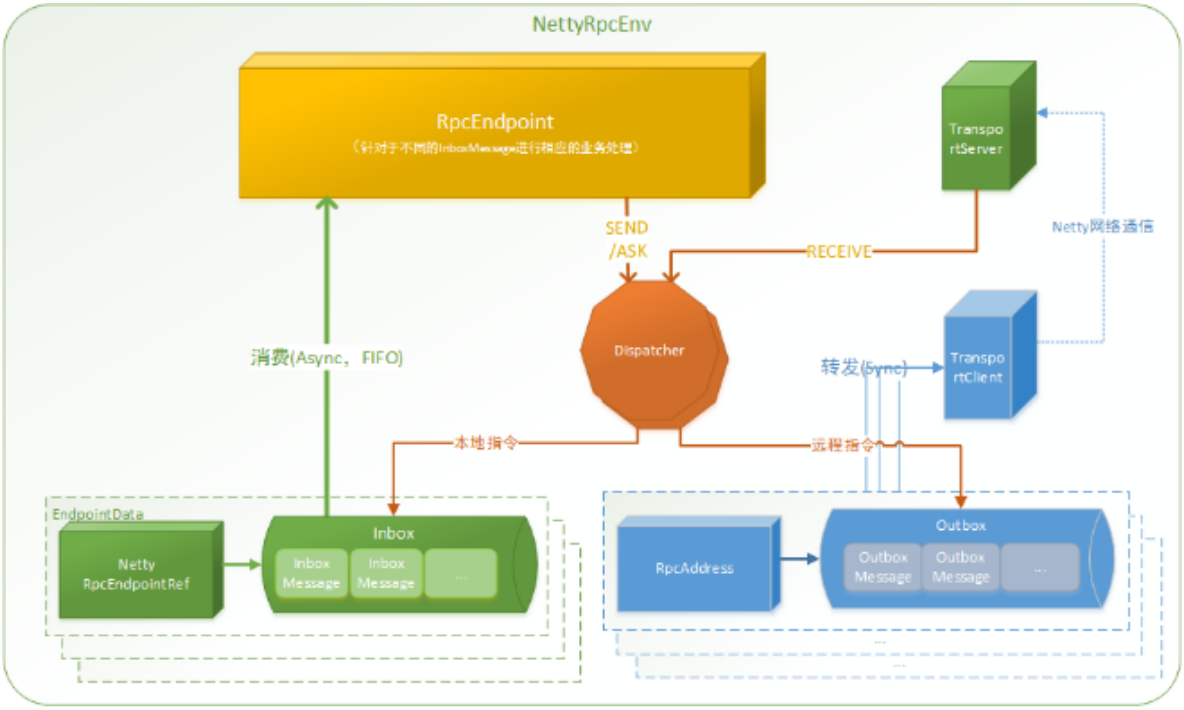
> Spark通讯框架中各个组件（Client/Master/Worker）可以认为是一个个独立的实体，各个实体之间通过消息来进行通信。具体各个组件之间的关系图如下：



> Endpoint（Client/Master/Worker）有1个InBox和N个OutBox（N>=1，N取决于当前Endpoint与多少其他的Endpoint进行通信，一个与其通讯的其他Endpoint对应一个OutBox），Endpoint接收到的消息被写入InBox，发送出去的消息写入OutBox并被发送到其他Endpoint的InBox中。

通讯架构解析

Spark通信架构如下图所示：



（1）RpcEndpoint：RPC端点。Spark针对每个节点（Client/Master/Worker）都称之为一个Rpc端点，且都实现RpcEndpoint接口，内部根据不同端点的需求，设计不同的消息和不同的业务处理，如果需要发送（询问）则调用Dispatcher；

（2） RpcEnv：RPC上下文环境，每个RPC端点运行时依赖的上下文环境称为RpcEnv；

（3）Dispatcher：消息分发器，针对于RPC端点需要发送消息或者从远程RPC接收到的消息，分发至对应的指令收件箱/发件箱。如果指令接收方是自己则存入收件箱，如果指令接收方不是自己，则放入发件箱；

（4）Inbox：指令消息收件箱。一个本地RpcEndpoint对应一个收件箱，Dispatcher在每次向Inbox存入消息时，都将对应EndpointData加入内部ReceiverQueue中，另外Dispatcher创建时会启动一个单独线程进行轮询ReceiverQueue，进行收件箱消息消费；

（5）RpcEndpointRef：RpcEndpointRef是对远程RpcEndpoint的一个引用。当我们需要向一个具体的RpcEndpoint发送消息时，一般我们需要获取到该RpcEndpoint的引用，然后通过该应用发送消息。

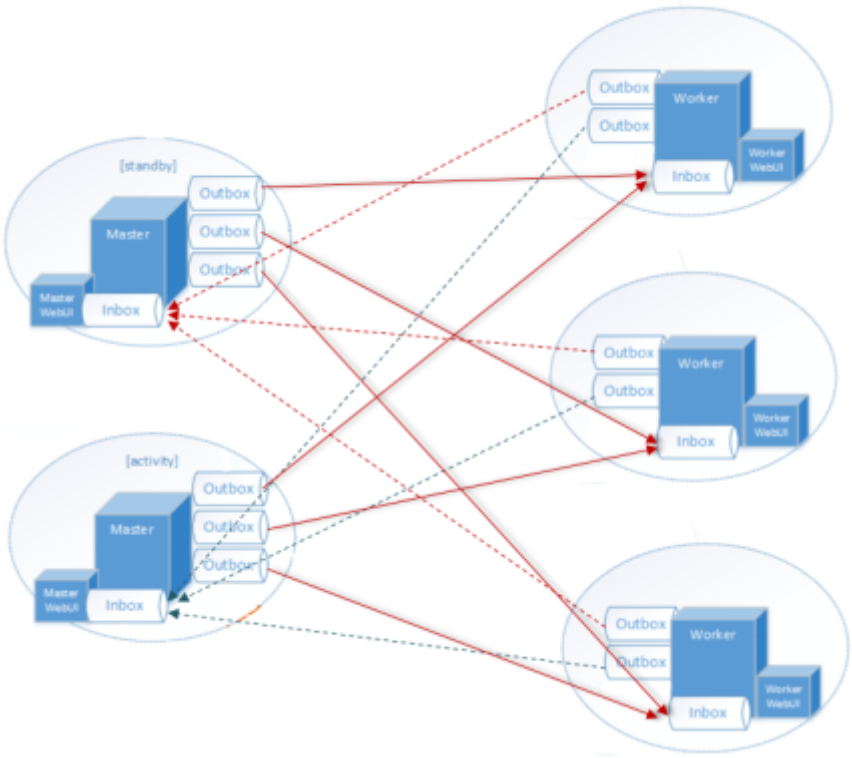
（6）OutBox：指令消息发件箱。对于当前RpcEndpoint来说，一个目标RpcEndpoint对应一个发件箱，如果向多个目标RpcEndpoint发送信息，则有多个OutBox。当消息放入Outbox后，紧接着通过TransportClient将消息发送出去。消息放入发件箱以及发送过程是在同一个线程中进行；

（7）RpcAddress：表示远程的RpcEndpointRef的地址，Host + Port。

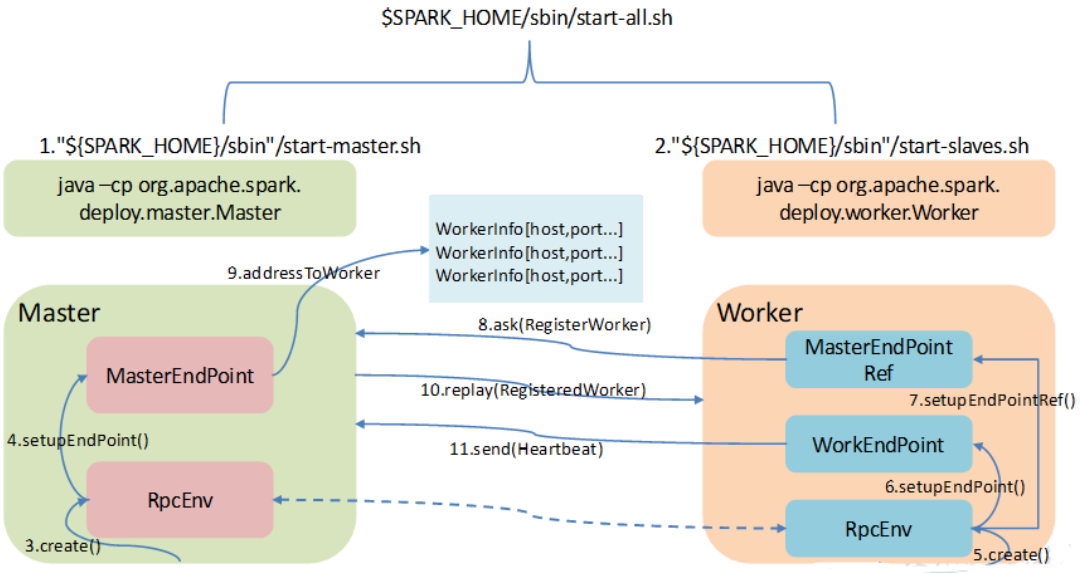
（8）TransportClient：Netty通信客户端，一个OutBox对应一个TransportClient，TransportClient不断轮询OutBox，根据OutBox消息的receiver信息，请求对应的远程TransportServer；

（9）TransportServer：Netty通信服务端，一个RpcEndpoint对应一个TransportServer，接受远程消息后调用Dispatcher分发消息至对应收发件箱；

根据上面的分析，Spark通信架构的高层视图如下图所示：



Spark集群启动【重点掌握】



（1）start-all.sh脚本，实际是执行“java -cp Master”和“java -cp Worker”；

（2）Master启动时首先创建一个RpcEnv对象，负责管理所有通信逻辑；

（3）Master通过RpcEnv对象创建一个Endpoint，Master就是一个Endpoint，Worker可以与其进行通信；

（4）Worker启动时也是创建一个RpcEnv对象；

（5）Worker通过RpcEnv对象创建一个Endpoint；

（6）Worker通过RpcEnv对象建立到Master的连接，获取到一个RpcEndpointRef对象，通过该对象可以与Master通信；

（7）Worker向Master注册，注册内容包括主机名、端口、CPU Core数量、内存数量；

（8）Master接收到Worker的注册，将注册信息维护在内存中的Table中，其中还包含了一个到Worker的RpcEndpointRef对象引用；

（9）Master回复Worker已经接收到注册，告知Worker已经注册成功；

（10）Worker端收到成功注册响应后，开始周期性向Master发送心跳。