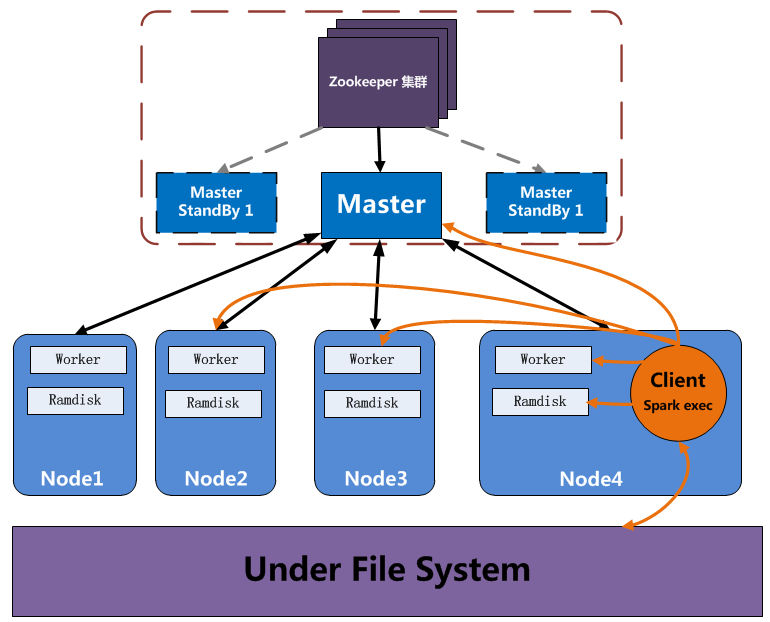
Alluxio RPC框架分析

Alluxio是主从架构，有主节点（AlluxioMaster）和多个从节点Worker(AlluxioWorker)构成，如下图所示：



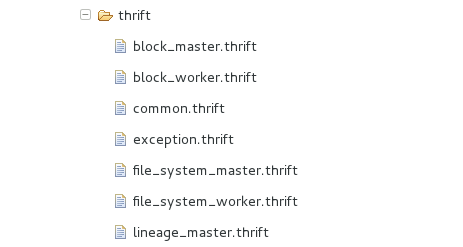
当Alluxio的Worker节点启动之后，Worker会向Master注册。注册成功之后，Master会与Worker之间维护一个心跳。Alluxio会将内存映射为Block设备，就是RamDisk。然后Alluxio会将数据存放在每个Worker的RamDisk（内存）中。Alluxio可以以内存的速度去读写Ramdisk中存放的数据。

# Alluxio RPC框架

Alluxio Master、Worker及Client之间的通信采用RPC通信模型，Alluxio的RPC是依靠Thrift实现，采用接口描述语言定义并创建服务，支持可扩展的跨语言服务开发，如下图所示：



Alluxio的RPC Thrift描述文件在$ALLUXIO\_HOME/core/common/alluxio/thirft目录下，描述文件如下所示：



* **common.thrift**，最基础的AlluxioService的类定义，包含了基本的通讯类型、块基本信息、通用的RPC基类，为其他RPC服务提供一些基础功能，如getServiceVersion
* **file\_system\_master.thirft**，定义Master端服务，实现包含两个Service类为：

FileSystemMasterClientService，Client至Master通信的重要方法，如create、createDiretory、completeFile、getStatus、LoadMetadata、mount等FileSystemMasterWokerService,Worker至Master的通信的重要方法，如getFileInfo,heartbeat等

* **file\_systm\_worker.thift**，定义Worker端服务FileSystemWorkerClientService，客户端到Worker节点的通讯方法，如createUfsFile,openUfsFile
* **block\_master.thirft，**定义Master端block管理服务BlockMasterWorkerService，每个Worker节点通过这个协议注册自己，并提交自己保存的块信息
* **block\_worker.thirft**，定义Worker端block管理服务BlockWorkerClientService，运行向当前Worke发起相关请求，如cancelBlock、lockBlock等相关方法
* **exception.thirft，**基础异常类
* **lineage\_master.thift，**文件之间依赖功能类，提高吞吐量（目前是试验版，不再介绍）

下表是Alluxio中六类RPC调用的服务类、处理Handler及通信方法：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Rpc Client | Rpc Service | Service Handler | 角色 | 方向 |
| FileSystemMasterClient | FileSystemMasterClientService | FileSystemMasterClientServiceHandler | FileSystemMaster | Client->Master |
| FileSystemMasterClient | FileSystemMasterWorkerService | FileSystemMasterWorkerServiceHandler | FileSystemMaster | Worker->Master |
| RetryHandlingBlockMasterClient | BlockMasterClientService | BlockMasterClientServiceHandler | BlockMaster | Client->Master |
| BlockMasterClient | BlockMasterWorkerService | BlockMasterClientServiceHandler | BlockMaster | Worker->Master |
| FileSystemWorkerClient | FileSystemWorkerClientService | FileSystemWorkerClientServiceHandler | FileSystemWorker | Client->Worker |
| RetryHandlingBlockWorkerClient | BlockWorkerClientService | BlockWorkerClientServiceHandler | BlockWorker | Client->Worker |

RPC客户端扩展AbstractThirftCleint类，使用方法retryRPC发起RPC调用，然后实现接口Service.Iface的抽象方法的ServiceHandler将会收到调用。

客户端通过ClientService发起RPC调用到实现RPC服务接口的ServiceHandler，调动的方向可以为：Client -> Master、Worker->Master和Client->Worker。

实现接口ClientService.Iface的ServiceHandler会被RPC客户端调用，通常ServiceHandler有一个成员来在Master或Worker上执行具体的操作。

# AlluxioMaster

AlluxioMaster是Alluxio的Master的实现，这个类中是AlluxioMaster启动的入口，其依赖的几个组件：

1. 数据块元数据管理服务（BlockManager）
2. 文件系统元数据信息管理服务（FileSystemManager）
3. Lineage元数据管理服务（LineageManager）
4. WebUI服务

AlluxioMaster启动时会以RPC Server的身份打开监听，等待客户端的连接，其相关类图如下所示：



AlluxioMaster的main函数启动后，通过AlluxioMasterProcess#start启动，代码如下：

*protected void startServing(String startMessage, String stopMessage) {*

*MetricsSystem.startSinks();*

*startServingWebServer();*

*startJvmMonitorProcess();*

*startServingRPCServer();*

*}*

startServingWebServer负责启动Web服务，startServingRPCServer方法负责启动BlockManager，FileSystemMaster及LineageMaser等服务。RpcServer的启动过程如下所示：

*protected void startServingRPCServer() {*

*TMultiplexedProcessor processor = new TMultiplexedProcessor();*

*for (Master master : mRegistry.getServers()) {*

*registerServices(processor, master.getServices());*

*}*

*// 注册Master Meta 服务*

*processor.registerProcessor(Constants.META\_MASTER\_SERVICE\_NAME,*

*new MetaMasterClientService.Processor<>(new MetaMasterClientServiceHandler(this)));*

*//打开Thrift Server的端口*

*TTransportFactory transportFactory;*

*try {*

*String serverName = NetworkAddressUtils.getConnectHost(ServiceType.MASTER\_RPC);*

*transportFactory = mTransportProvider.getServerTransportFactory(serverName);*

*} catch (IOException e) {*

*throw new RuntimeException(e);*

*}*

*try {*

*if (mTServerSocket != null) {*

*mTServerSocket.close();*

*}*

*mTServerSocket = new TServerSocket(mRpcBindAddress,*

*(int) Configuration.getMs(PropertyKey.MASTER\_CONNECTION\_TIMEOUT\_MS));*

*} catch (TTransportException e) {*

*throw new RuntimeException(e);*

*}*

*//向Thrift Server Processor中添加对外提供的RPC Thirft服务*

*//包括的Thrift 服务：*

*—FileSystemMasterClientService*

*—FileSystemMasterWorkerService*

*—BlockMasterWorkerService*

*—BlockMasterClientService*

*Args args = new TThreadPoolServer.Args(mTServerSocket).maxWorkerThreads(mMaxWorkerThreads)*

*.minWorkerThreads(mMinWorkerThreads).processor(processor).transportFactory(transportFactory)*

*.protocolFactory(new TBinaryProtocol.Factory(true, true));*

*args.stopTimeoutVal = (int) Configuration.getMs(PropertyKey.MASTER\_THRIFT\_SHUTDOWN\_TIMEOUT);*

*mThriftServer = new TThreadPoolServer(args);*

*//启动Thrift RPC Server*

*mIsServing = true;*

*mStartTimeMs = System.currentTimeMillis();*

*mSafeModeManager.notifyRpcServerStarted();*

*mThriftServer.serve();*

*}*

# AlluxioWorker

AlluxioWorker提供本地管理服务（BlockWorker）、文件系统管理（FileSystemWorker）和数据请求服务（DataServer）及对外提供的节点RPC服务，类图如下所示：



AlluxioWorkerProcess#start过程如下：

*@Override*

*public void start() throws Exception {*

*MetricsSystem.startSinks();*

*startWorkers(); //启动各RPC 服务*

*mWebServer.addHandler(mMetricsServlet.getHandler());*

*mWebServer.start(); //启动Web服务*

*if (Configuration.getBoolean(PropertyKey.WORKER\_JVM\_MONITOR\_ENABLED)) {*

*mJvmPauseMonitor = new JvmPauseMonitor();*

*mJvmPauseMonitor.start();*

*}*

*mIsServingRPC = true;*

*mThriftServer.serve(); //启动Thrift Server*

*}*

# Alluxio Client

Alluxio Client实现与Server之间交互通过RPC来实现，以FileSystemMasterClientService为例，Client端的实现是在FileSystemMasterClient类中，其内部有一个FileSystemMaster-

ClientService.Client类型的成员变量，其如下所示：

*FileSystemMasterClientService.Client mClient = null*

其初始化在afterConnection中完成，如下所示：

*@Override*

*protected void afterConnect() throws IOException {*

*mClient = new FileSystemMasterClientService.Client(mProtocol);*

*}*

其使用过程，以读文件getStatus方法为例：

*public synchronized URIStatus getStatus(final AlluxioURI path, final GetStatusOptions options)*

*throws AlluxioStatusException {*

*return retryRPC(() -> new URIStatus(ThriftUtils*

*.fromThrift(mClient.getStatus(path.getPath(), options.toThrift()).getFileInfo())));*

*}*

执行retryRpc方法中，发起一个RPC调用，然后执行call方法构造一个URIStatus实例，并返回，构造时需要使用FileInfo对象，其通过FileSystemMasterClientService的getStatus方法调用，也就是调用服务端的getStatus方法。retryRpc方法，定义在AbstractClient中，如下所示：

*protected synchronized <V> V retryRPC(RpcCallable<V> rpc) throws AlluxioStatusException {*

*RetryPolicy retryPolicy =*

*ExponentialTimeBoundedRetry.builder().withMaxDuration(MAX\_RETRY\_DURATION)*

*.withInitialSleep(BASE\_SLEEP\_MS).withMaxSleep(MAX\_SLEEP\_MS).build();*

*Exception ex = null;*

*while (retryPolicy.attempt()) {*

*if (mClosed) {*

*throw new FailedPreconditionException("Client is closed");*

*}*

*connect(); //进行thrift rpc连接*

*try {*

*return rpc.call(); //调用方法*

*}*

*}*

其核心是connect和call方法，connect方法

*public synchronized void connect() throws AlluxioStatusException {*

*while (retryPolicy.attempt()) {*

*try {*

*mAddress = getAddress(); //1.获取master的地址mAddress*

*} .....*

*mProtocol = TProtocols.createProtocol( //2.创建传输协议TbinaryProtocol，二进制协议*

*mTransportProvider.getClientTransport(mParentSubject, mAddress), getServiceName());*

*try {*

*mProtocol.getTransport().open(); //3.启动rpc连接*

*mConnected = true; //4.标识连接已经建立*

*afterConnect(); //5.创建client实例*

*checkVersion(getClient(), getServiceVersion());*

*return;*

*}*

*}*

*}*

RPC 连接一旦建立，就可以象调用本地方法进行远程调用，调用getStatus获取结果后利用ThiftUtils的数据转换方法fromThrift进行对象转换，得到需要的对象数据。

http://www.iwantfind.com/thread-343-1-1.html

https://hacpai.com/article/1484726852429?p=1&m=0

https://blog.csdn.net/lipeng\_bigdata/article/details/50978732