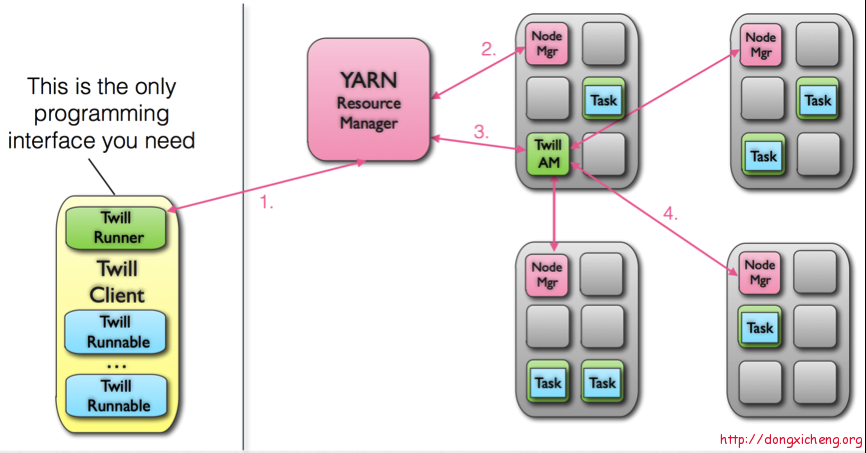
Apache Twill

Apache Twill是为了简化YARN上应用程序开发而成立的项目，该项目把YARN相关的重复性工作封装成库，使得用户可以专注于自己的应用程序逻辑。Twill所在的事情就是开发一个通用的Client和AppMaster实现，并提供一套编程API，使得用户可直接实现这套API，便可以编写运行在YARN上的分布式程序。

Twill应用程序运行在YARN上的流程和普通程序一样，流程如下所示：



为极大方便用户编写运行在YARN上的分布式应用程序，Twill做了大量通用模块和特性，包括：

1. 实时日志收集，基于kafka实现日志收集功能，可将应用程序产生的日志实时收集起来，便于集中化分析和展示
2. 资源汇报，可实时收集任务占用的各类资源，包括cpu利用率，物理内存等
3. 资源弹性扩展，可动态改变应用程序占用的资源，比如将container数目在线由5个增长为10个
4. 状态恢复，当客户端失去与应用程序连接后，可通过API重构一个新的客户端，重新获取状态
5. 服务发现，可向zookeeper注册一些信息，便于其他服务或者客户端发现该应用程序的位置，以便与之通信

# Apache Twill的使用

在Twill源码中，有两个example可以运行，向日志中输出一行log。可以通过下面命令在Hadoop 集群中启动应用：

*export CP=twill-examples-yarn-0.12.0-SNAPSHOT.jar:`hadoop classpath`*

*export HADOOP\_HOME=/cmss/bch/bc1.3.5/hadoop*

*java -cp $CP org.apache.twill.example.yarn.HelloWorld fys1.cmss.com:2181*

运行后，在命令终端中输出日志：

*INFO o.a.t.i.ServiceMain [fys1.cmss.com] [main]ServiceMain:doMain(ServiceMain.java:94) - Service TwillContainerService [RUNNING] started.*

*INFO o.a.t.e.y.HelloWorld [fys1.cmss.com] [TwillContainerService] HelloWorld$HelloWorldRunnable:run(HelloWorld.java:52) - Hello World. My first distributed application.*

BundleJarExample，支持用户在Twill中运行应用程序时不用担心应用与Hadoop之间的lib版本冲突问题。该例子仅简单的调用Echo应用，该应用输出命令行中输出的日志。Echo应用中使用与Twill及Hadoop不同的Guava版本。BundledJarExample依赖Echo的lib文件夹。运行命令如下所示：

*java -cp $CP org.apache.twill.example.yarn.BundledJarExample fys1.cmss.com:2181 twill-examples-echo-0.12.0-SNAPSHOT.jar echo.EchoMain arg1*

# Apache Twill源码分析

Apache Twill的执行流程如下所示：



## 2.1 Twill Driver

启动Twill 应用的客户端，其核心为TwillRunnerServer和TwillController，初始化Twill的资源配置和向集群提交应用创建请求。实例中的应用启动的初始代码如下：

*TwillRunnerService twillRunner = new YarnTwillRunnerService(yarnConfiguration, zkStr);*

*twillRunner.start();*

*....*

*final TwillController controller = twillRunner.prepare(new HelloWorldRunnable())*

*.addLogHandler(new PrinterLogHandler(new PrintWriter(System.out, true)))*

*.withApplicationClassPaths(applicationClassPaths)*

*.withBundlerClassAcceptor(new HadoopClassExcluder())*

*.start();*

其中TwillRunnerServer.prepare该方法用于初始化Twill应用相关参数，其描述类为TwillSpecification，其生成如下：

*public TwillPreparer prepare(TwillApplication application) {*

*final TwillSpecification twillSpec = application.configure();*

*return new YarnTwillPreparer(config, twillSpec, runId, zkClientService.getConnectString(),*

*appLocation, jvmOptions, locationCache,*

*new YarnTwillControllerFactory() {*

*@Override*

*public YarnTwillController create(RunId runId, boolean logCollectionEnabled, Iterable<LogHandler> logHandlers,*

*Callable<ProcessController<YarnApplicationReport>> startUp,*

*long startTimeout, TimeUnit startTimeoutUnit) {*

*......*

*}*

*return controller;*

*}*

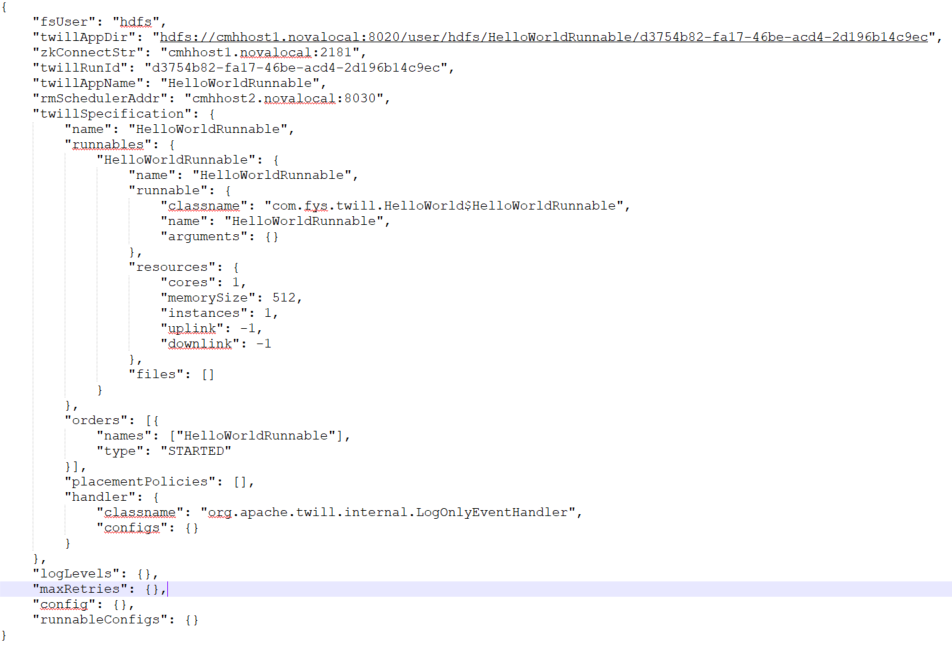
*});*

*}*

相关类如下：



TwillSpecification类会持久化到Container执行目录的Json文件，作为Container执行的参数，示例如下：



## **ApplicationMaster**

Driver向RM申请资源，并启动ApplicationMaster，其用于管理实际执行Container，其作用不再详述，其启动类为ApplicationMasterProcessLauncher，执行如下：

YarnTwillPreparer#start

*@Override*

*public TwillController start(long timeout, TimeUnit timeoutUnit) {*

*try {*

*....*

*final ProcessLauncher<ApplicationMasterInfo> launcher = yarnAppClient.createLauncher(twillSpec, schedulerQueue);*

*final ApplicationMasterInfo appMasterInfo = launcher.getContainerInfo();*

*.....*

*twillRuntimeSpec = saveSpecification(twillSpec, runtimeConfigDir.resolve(Constants.Files.TWILL\_SPEC));*

*.....*

*return launcher.prepareLaunch(ImmutableMap.<String, String>of(), localFiles.values(),*

*createSubmissionCredentials())*

*.addCommand(*

*"$JAVA\_HOME/bin/java",*

*"-Djava.io.tmpdir=tmp",*

*"-Dyarn.appId=$" + EnvKeys.YARN\_APP\_ID\_STR,*

*"-Dtwill.app=$" + Constants.TWILL\_APP\_NAME,*

*"-cp", Constants.Files.LAUNCHER\_JAR + ":$HADOOP\_CONF\_DIR",*

*"-Xmx" + memory + "m",*

*jvmOptions.getAMExtraOptions(),*

*TwillLauncher.class.getName(),*

*ApplicationMasterMain.class.getName(),*

*Boolean.FALSE.toString())*

*.launch();*

*}*

*};*

启动ApplicationMasterMain类，并将TwillRuntimeSpecification保存到文件中。在ApplicationMaster中，其中核心成员变量为ApplicationMasterService，该成员用于生成Container请求并申请资源

1. 初始化Container请求，在Task Container中启动具体的任务(TwillRunnable)

*private Queue<RunnableContainerRequest> initContainerRequests() {*

*Queue<RunnableContainerRequest> requests = new ConcurrentLinkedQueue<>();*

*//根据分配Policy及启动顺序，将可以启动的Container，添加到队列中*

*for (TwillSpecification.Order order : twillSpec.getOrders()) {*

*Set<String> distributedRunnables = Sets.intersection(placementPolicyManager.getDistributedRunnables(),*

*....*

*requests.add(new RunnableContainerRequest(order.getType(), requestsMap));*

*}*

*return requests;*

*}*

1. 当靠前的Container执行完以后，将依赖其执行的Container请求添加到Request队列中，在handleCompleted中调用createRunnableContainerRequest，其执行如下：

*private RunnableContainerRequest createRunnableContainerRequest(final String runnableName,*

*final int numberOfInstances, final boolean isProvisioned) {*

*//获取启动顺序*

*TwillSpecification.Order order = Iterables.find(twillSpec.getOrders(),*

*new Predicate<TwillSpecification.Order>() {*

*@Override*

*public boolean apply(TwillSpecification.Order input) {*

*return (input.getNames().contains(runnableName));*

*}*

*});*

*//获取要启动的Runnable类的配置*

*RuntimeSpecification runtimeSpec = twillSpec.getRunnables().get(runnableName);*

*//添加到队列中*

*addAllocationSpecification(allocationSpecification, requestsMap, runtimeSpec);*

*}*

*return new RunnableContainerRequest(order.getType(), requestsMap, isProvisioned);*

*}*

## **2.3 TwillContainer的执行**

ApplicationMaster获取Container资源后启动TwillContainer，其启动如下：

*TwillContainerLauncher launcher = new TwillContainerLauncher(*

*twillSpec.getRunnables().get(runnableName), processLauncher.getContainerInfo(), launchContext, ZKClients.namespace(zkClient, getZKNamespace(runnableName)),*

*containerCount, jvmOpts, twillRuntimeSpec.getReservedMemory(runnableName),*

*twillRuntimeSpec.getMinHeapRatio(runnableName), getSecureStoreLocation());*

*runningContainers.start(runnableName, processLauncher.getContainerInfo(), launcher);*

其中Container根据相关的RuntimeSpecification来执行，TwillContainerMain的执行如下：

*public static void main(String[] args) throws Exception {  
 File twillSpecFile = new File(Constants.Files.RUNTIME\_CONFIG\_JAR,*

*Constants.Files.TWILL\_SPEC);  
 TwillRuntimeSpecification twillRuntimeSpec = TwillRuntimeSpecificationAdapter.create().fromJson(twillSpecFile);  
 new TwillContainerMain(twillRuntimeSpec).doMain();  
}*

执行如下：

*private void doMain() throws Exception {*

*//获取Runnable类名*

*String runnableName = System.getenv(EnvKeys.TWILL\_RUNNABLE\_NAME);*

*//获取向对应的配置文件*

*TwillRunnableSpecification runnableSpec = twillRuntimeSpec.getTwillSpecification().getRunnables().get(runnableName).getRunnableSpecification();*

*//解析参数*

*Arguments arguments = decodeArgs();*

*BasicTwillContext context = new BasicTwillContext(*

*runnableSpec*

*arguments*

*......);*

*//启动TwillContainerService*

*TwillContainerService service = new TwillContainerService(context, ....*

*doMain( service, ......);*

*}*

TwillRunnable的调用执行在TwillContainerService中，如下所示：

*@SuppressWarnings("unchecked")  
@Override  
protected void doStart() throws Exception {*

*///命令执行线程  
 commandExecutor = Executors.newSingleThreadExecutor(  
 Threads.createDaemonThreadFactory("runnable-command-executor"));  
 //初始化runnableClass  
 Class<?> runnableClass = classLoader.loadClass(specification.getClassName());  
 Preconditions.checkArgument(TwillRunnable.class.isAssignableFrom(runnableClass),  
 .., specification.getClassName());  
 runnable = Instances.newInstance((Class<TwillRunnable>) runnableClass); //核心处  
 runnable.initialize(context);  
}  
  
@Override  
protected void doRun() throws Exception {  
 runnable.run();  
}*