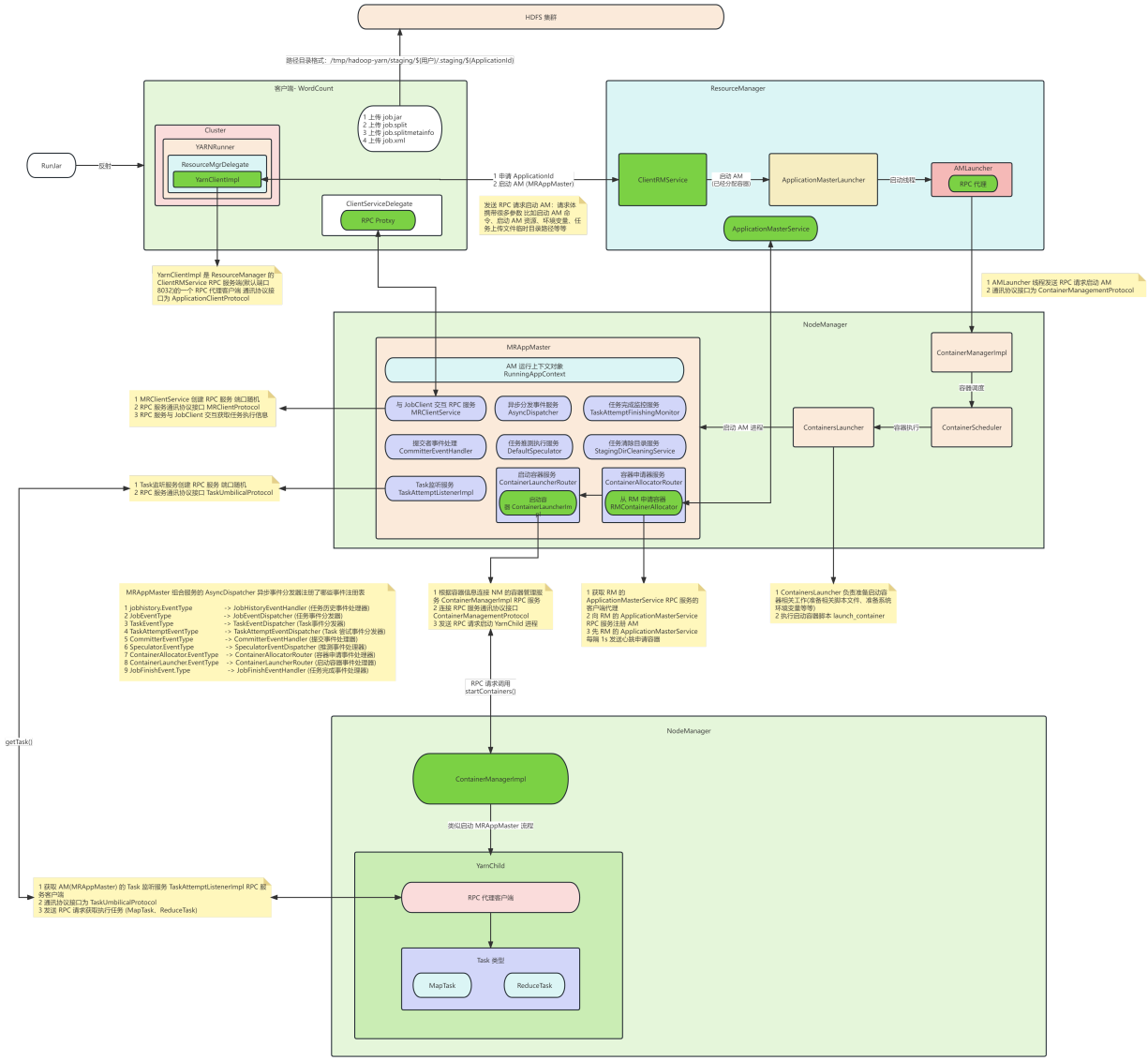
mapreduce-on-yarn任务提交源码分析



一 官方 wordcount 任务执行案例

```
# 一般企业提交 MR 任务到 YARN 集群基于 hadoop jar 方式
hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-examples-3.1.3.jar wordcount
/wordcount/input /wordcount/output
```

二 提交任务入口类

2.1 基于 hadoop/yarn jar 命令提交 (RunJar)

```
# hadoop.sh/yarn.sh
jar)
      if [[ -n "${YARN_OPTS}" ]] || [[ -n "${YARN_CLIENT_OPTS}" ]]; then
        hadoop_error "WARNING: Use \"yarn jar\" to launch YARN applications."
      fi
      if [[ -z $1 || $1 = "--help" ]]; then
        echo "Usage: hadoop jar <jar> [mainClass] args..."
        exit 0
      fi
      HADOOP_CLASSNAME=org.apache.hadoop.util.RunJar
    ;;
```

2.2 基于 mapred jar 命令提交 (JobClient)

```
# mapred.sh
job)
      HADOOP_CLASSNAME=org.apache.hadoop.mapred.JobClient
```

2.3 使用 RunJar 作为入口类

```
/** Run a Hadoop job jar. */
@InterfaceAudience.Private
@InterfaceStability.Unstable
public class RunJar {
   /** Run a Hadoop job jar.  If the main class is not in the jar's manifest,
   * then it must be provided on the command line. */
  public static void main(String[] args) throws Throwable {
    new RunJar().run(args);
  }
}
```

```
public void run(String[] args) throws Throwable {
        String usage = "RunJar jarFile [mainClass] args...";

        if (args.length < 1) {
            System.err.println(usage);
            System.exit(-1);
        }

        int firstArg = 0;
        // 判断入口 jar 是否存在
        String fileName = args[firstArg++];
        File file = new File(fileName);
        if (!file.exists() || !file.isFile()) {
            System.err.println("JAR does not exist or is not a normal file: " +
                    file.getCanonicalPath());
            System.exit(-1);
        }
        String mainClassName = null;
```

```
JarFile jarFile;
try {
    // 读取 jar 的启动类
    jarFile = new JarFile(fileName);
} catch (IOException io) {
    throw new IOException("Error opening job jar: " + fileName)
            .initCause(io);
}
Manifest manifest = jarFile.getManifest();
if (manifest != null) {
    mainClassName = manifest.getMainAttributes().getValue("Main-Class");
}
jarFile.close();

if (mainClassName == null) {
    if (args.length < 2) {
        System.err.println(usage);
        System.exit(-1);
    }
    mainClassName = args[firstArg++];
}
mainClassName = mainClassName.replaceAll("/", ".");

File tmpDir = new File(System.getProperty("java.io.tmpdir"));
ensureDirectory(tmpDir);

final File workDir;
try {
    workDir = File.createTempFile("hadoop-unjar", "", tmpDir);
} catch (IOException ioe) {
    // If user has insufficient perms to write to tmpDir, default
    // "Permission denied" message doesn't specify a filename.
    System.err.println("Error creating temp dir in java.io.tmpdir "
            + tmpDir + " due to " + ioe.getMessage());
    System.exit(-1);
    return;
}

if (!workDir.delete()) {
    System.err.println("Delete failed for " + workDir);
    System.exit(-1);
}
ensureDirectory(workDir);

ShutdownHookManager.get().addShutdownHook(
        new Runnable() {
            @Override
            public void run() {
                FileUtil.fullyDelete(workDir);
            }
        }, SHUTDOWN_HOOK_PRIORITY);
```

```
        unJar(file, workDir);

        // 创建类加载器并替换默认当前线程的类加载器
        ClassLoader loader = createClassLoader(file, workDir);
        Thread.currentThread().setContextClassLoader(loader);

        // 反射获取任务启动类的 main()
        Class<?> mainClass = Class.forName(mainClassName, true, loader);
        Method main = mainClass.getMethod("main", String[].class);
        List<String> newArgsSubList = Arrays.asList(args)
                .subList(firstArg, args.length);
        String[] newArgs = newArgsSubList
                .toArray(new String[newArgsSubList.size()]);
        try {
            // 调用任务启动类的 main() (比如 WordCount.main())
            main.invoke(null, new Object[]{newArgs});
        } catch (InvocationTargetException e) {
            throw e.getTargetException();
        }
    }
```

三 任务启动类入口

```
public class WordCount {

    public static class TokenizerMapper
            extends Mapper<Object, Text, Text, IntWritable> {

        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();

        public void map(Object key, Text value, Context context
        ) throws IOException, InterruptedException {
            StringTokenizer itr = new StringTokenizer(value.toString());
            while (itr.hasMoreTokens()) {
                word.set(itr.nextToken());
                context.write(word, one);
            }
        }
    }

    public static class IntSumReducer
            extends Reducer<Text, IntWritable, Text, IntWritable> {
        private IntWritable result = new IntWritable();

        public void reduce(Text key, Iterable<IntWritable> values,
                           Context context
        ) throws IOException, InterruptedException {
            int sum = 0;
```

```
            for (IntWritable val : values) {
                sum += val.get();
            }
            result.set(sum);
            context.write(key, result);
        }
    }

    public static void main(String[] args) throws Exception {
        // 创建 Configuration 并解析入口参数
        Configuration conf = new Configuration();
        String[] otherArgs = new GenericOptionsParser(conf, args)
                .getRemainingArgs();
        if (otherArgs.length < 2) {
            System.err.println("Usage: wordcount <in> [<in>...] <out>");
            System.exit(2);
        }
        // MapReduce 程序的标配
        Job job = Job.getInstance(conf, "word count");
        job.setJarByClass(WordCount.class);
        job.setMapperClass(TokenizerMapper.class);
        job.setCombinerClass(IntSumReducer.class);
        job.setReducerClass(IntSumReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        // WordCount 的输出输出路径
        for (int i = 0; i < otherArgs.length - 1; ++i) {
            FileInputFormat.addInputPath(job, new Path(otherArgs[i]));
        }
        FileOutputFormat.setOutputPath(job,
                new Path(otherArgs[otherArgs.length - 1]));

        // 执行任务等待完成
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

```
/**
     * Submit the job to the cluster and wait for it to finish.
     *
     * @param verbose print the progress to the user
     * @return true if the job succeeded
     * @throws IOException thrown if the communication with the
     *                     <code>JobTracker</code> is lost
     */
    public boolean waitForCompletion(boolean verbose
    ) throws IOException, InterruptedException,
            ClassNotFoundException {
        // state 初始化状态为 DEFINE
        if (state == JobState.DEFINE) {
```

```
            // 提交任务
            submit();
        }
        // 是否打印任务进度给用户
        if (verbose) {
            monitorAndPrintJob();
        } else {
            // get the completion poll interval from the client.
            int completionPollIntervalMillis =
                    Job.getCompletionPollInterval(cluster.getConf());
            while (!isComplete()) {
                try {
                    Thread.sleep(completionPollIntervalMillis);
                } catch (InterruptedException ie) {
                }
            }
        }
        return isSuccessful();
    }
```

```
/**
     * Submit the job to the cluster and return immediately.
     *
     * @throws IOException
     */
    public void submit()
            throws IOException, InterruptedException, ClassNotFoundException {
        // 确保任务状态 state 为 DEFINE
        ensureState(JobState.DEFINE);

        // 设置任务使用新 API
        setUseNewAPI();

        // 连接 ResourceManager 获取 ClientRMService 代理 RPC 客户端
        /**
         * 1 创建 Cluster 其构造方法 创建 YARNRunner 服务
         * 2 YARNRunner 服务的构造方法创建 ResourceMgrDelegate 服务
         * 3 ResourceMgrDelegate 服务的构造方法创建 YarnClientImpl 服务
         * 4.1 ResourceMgrDelegate 服务的构造方法调用其 serviceInit() 进而调用
YarnClientImpl.serviceInit()
         *     YarnClientImpl.serviceInit() 从配置文件获取对应的信息
         * 4.2 ResourceMgrDelegate 服务的构造方法调用其 serviceStart() 进而调用
YarnClientImpl.serviceStart()
         *     获取 ResourceManager 组合服务的子服务 ClientRMService RPC Server (端口
8032)
         *     的代理对象 RPC 通讯协议接口为 ApplicationClientProtocol
         */
        connect();

        // 获取任务提交对象 JobSubmitter
        final JobSubmitter submitter =
```

```
                getJobSubmitter(
                        cluster.getFileSystem(),
                        cluster.getClient());
        status = ugi.doAs(new PrivilegedExceptionAction<JobStatus>() {
            public JobStatus run() throws IOException, InterruptedException,
                    ClassNotFoundException {
                // 提交任务
                return submitter.submitJobInternal(Job.this, cluster);
            }
        });
        state = JobState.RUNNING;
        // The url to track the job:
http://hadoop103:8088/proxy/application_1684656010852_0001/
        LOG.info("The url to track the job: " + getTrackingURL());
    }
```

3.1 连接 ResourceManager 获取 ClientRMService 代理 RPC 客户端

```
    private synchronized void connect()
            throws IOException, InterruptedException, ClassNotFoundException {
        if (cluster == null) {
            cluster =
                    ugi.doAs(new PrivilegedExceptionAction<Cluster>() {
                        public Cluster run()
                                throws IOException, InterruptedException,
                                ClassNotFoundException {
                            // 创建 Cluster
                            return new Cluster(getConfiguration());
                        }
                    });
        }
    }
```

3.1.1 创建 Cluster

```
/**
 * Provides a way to access information about the map/reduce cluster.
 */
@InterfaceAudience.Public
@InterfaceStability.Evolving
public class Cluster {
     public Cluster(Configuration conf) throws IOException {
        // 往下追
        this(null, conf);
    }
}
```

```
    public Cluster(InetSocketAddress jobTrackAddr, Configuration conf)
            throws IOException {
        this.conf = conf;
        this.ugi = UserGroupInformation.getCurrentUser();
        // 初始化 Cluster
        initialize(jobTrackAddr, conf);
    }
```

```
private void initialize(InetSocketAddress jobTrackAddr, Configuration conf)
            throws IOException {

        // 加载 ClientProtocolProvider 所有子类
        // 1 LocalClientProtocolProvider (本地)
        // 2 YarnClientProtocolProvider (yarn)
        initProviderList();
        final IOException initEx = new IOException(
                "Cannot initialize Cluster. Please check your configuration for "
                        + MRConfig.FRAMEWORK_NAME
                        + " and the correspond server addresses.");
        // jobTrackAddr 默认 null
        if (jobTrackAddr != null) {
            LOG.info(
                    "Initializing cluster for Job Tracker=" +
jobTrackAddr.toString());
        }
        for (ClientProtocolProvider provider : providerList) {
            LOG.debug("Trying ClientProtocolProvider : "
                    + provider.getClass().getName());
            ClientProtocol clientProtocol = null;
            try {
                if (jobTrackAddr == null) {
                    // 根据 key = mapreduce.framework.name value = yarn (一般在
mapred-site.xml 配置)
                    // 故直接调用 YarnClientProtocolProvider.create() 返回 YARNRunner
                    clientProtocol = provider.create(conf);
                } else {
                    clientProtocol = provider.create(jobTrackAddr, conf);
                }

                if (clientProtocol != null) {
                    // 基于 yarn 方式提交任务 clientProtocolProvider =
YarnClientProtocolProvider
                    clientProtocolProvider = provider;
                    // YARNRunner
                    client = clientProtocol;
                    LOG.debug("Picked " + provider.getClass().getName()
                            + " as the ClientProtocolProvider");
                    break;
                } else {
                    LOG.debug("Cannot pick " + provider.getClass().getName()
```

```
                            + " as the ClientProtocolProvider - returned null
protocol");
                }
            } catch (Exception e) {
                final String errMsg = "Failed to use " +
provider.getClass().getName()
                        + " due to error: ";
                initEx.addSuppressed(new IOException(errMsg, e));
                LOG.info(errMsg, e);
            }
        }

        if (null == clientProtocolProvider || null == client) {
            throw initEx;
        }
    }
```

3.1.1.1 创建 YARNRunner

```
// 调用 YarnClientProtocolProvider.create()
public class YarnClientProtocolProvider extends ClientProtocolProvider {

  @Override
  public ClientProtocol create(Configuration conf) throws IOException {
    if (MRConfig.YARN_FRAMEWORK_NAME.equals(conf.get(MRConfig.FRAMEWORK_NAME))) {
      // 创建 YARNRunner
      return new YARNRunner(conf);
    }
    return null;
  }
}
```

```
/**
 * This class enables the current JobClient (0.22 hadoop) to run on YARN.
 */
@SuppressWarnings("unchecked")
public class YARNRunner implements ClientProtocol {
 /**
     * Yarn runner incapsulates the client interface of
     * yarn
     *
     * @param conf the configuration object for the client
     */
    public YARNRunner(Configuration conf) {
        // 往下追
        this(conf,
                // 创建资源管理委托服务 ResourceMgrDelegate (与 ResourceManager 打交道)
                new ResourceMgrDelegate(new YarnConfiguration(conf))
        );
    }
}
```

3.1.1.1.1 创建 ResourceMgrDelegate(真正 RPC 客户端 YarnClientImpl)

```
public class ResourceMgrDelegate extends YarnClient {
 /**
     * Delegate responsible for communicating with the Resource Manager's
     * {@link ApplicationClientProtocol}.
     *
     * @param conf the configuration object.
     */
    public ResourceMgrDelegate(YarnConfiguration conf) {
        super(ResourceMgrDelegate.class.getName());
        this.conf = conf;
        // 创建 Yarn 客户端实例服务 YarnClientImpl
        this.client = YarnClient.createYarnClient();
        // 初始化 (调用 this.serviceInit())
        init(conf);
        // 启动 (调用 this.serviceStart())
        start();
    }
}
```

```
    @Override
    protected void serviceInit(Configuration conf) throws Exception {
        // 初始化 Yarn 客户端实例服务 (调用 YarnClientImpl.serviceInit())
        client.init(conf);
        super.serviceInit(conf);
    }
```

```
    @Override
    protected void serviceStart() throws Exception {
        // 启动 Yarn 客户端实例服务 (调用 YarnClientImpl.serviceStart())
        client.start();
        super.serviceStart();
    }
```

```
// 调用 YarnClientImpl.serviceInit()
public class YarnClientImpl extends YarnClient {
    @Override
    protected void serviceInit(Configuration conf) throws Exception {
        // 获取相关配置信息

        // 默认 200ms
        asyncApiPollIntervalMillis =

 conf.getLong(YarnConfiguration.YARN_CLIENT_APPLICATION_CLIENT_PROTOCOL_POLL_INTERVA
L_MS,

 YarnConfiguration.DEFAULT_YARN_CLIENT_APPLICATION_CLIENT_PROTOCOL_POLL_INTERVAL_MS)
;
        asyncApiPollTimeoutMillis =
```

```
 conf.getLong(YarnConfiguration.YARN_CLIENT_APPLICATION_CLIENT_PROTOCOL_POLL_TIMEOUT
_MS,

 YarnConfiguration.DEFAULT_YARN_CLIENT_APPLICATION_CLIENT_PROTOCOL_POLL_TIMEOUT_MS);
        submitPollIntervalMillis = asyncApiPollIntervalMillis;
        if (conf.get(YarnConfiguration.YARN_CLIENT_APP_SUBMISSION_POLL_INTERVAL_MS)
                != null) {
            submitPollIntervalMillis = conf.getLong(
                    YarnConfiguration.YARN_CLIENT_APP_SUBMISSION_POLL_INTERVAL_MS,

 YarnConfiguration.DEFAULT_YARN_CLIENT_APPLICATION_CLIENT_PROTOCOL_POLL_INTERVAL_MS)
;
        }

        if (YarnConfiguration.timelineServiceV1Enabled(conf)) {
            timelineV1ServiceEnabled = true;
            timelineDTRenewer = getTimelineDelegationTokenRenewer(conf);
            timelineService = TimelineUtils.buildTimelineTokenService(conf);
        }

        if (YarnConfiguration.timelineServiceV2Enabled(conf)) {
            timelineV2ServiceEnabled = true;
        }

        // The AHSClientService is enabled by default when we start the
        // TimelineServer which means we are able to get history information
        // for applications/applicationAttempts/containers by using ahsClient
        // when the TimelineServer is running.
        if (timelineV1ServiceEnabled || conf.getBoolean(
                YarnConfiguration.APPLICATION_HISTORY_ENABLED,
                YarnConfiguration.DEFAULT_APPLICATION_HISTORY_ENABLED)) {
            historyServiceEnabled = true;
            historyClient = AHSClient.createAHSClient();
            historyClient.init(conf);
        }

        if (timelineV2ServiceEnabled) {
            ahsV2Client = AHSClient.createAHSv2Client();
            ahsV2Client.init(conf);
        }

        timelineServiceBestEffort = conf.getBoolean(
                YarnConfiguration.TIMELINE_SERVICE_CLIENT_BEST_EFFORT,
                YarnConfiguration.DEFAULT_TIMELINE_SERVICE_CLIENT_BEST_EFFORT);

        loadResourceTypesFromServer = conf.getBoolean(
                YarnConfiguration.YARN_CLIENT_LOAD_RESOURCETYPES_FROM_SERVER,

 YarnConfiguration.DEFAULT_YARN_CLIENT_LOAD_RESOURCETYPES_FROM_SERVER);

        super.serviceInit(conf);
```

```
    }
}
```

```
// 调用 YarnClientImpl.serviceStart()
@Override
    protected void serviceStart() throws Exception {
        try {
            // 获取 ResourceManager 组合服务的子服务 ClientRMService RPC Server (端口
8032)
            // 的代理对象 RPC 通讯协议接口为 ApplicationClientProtocol
            rmClient = ClientRMProxy.createRMProxy(getConfig(),
                    ApplicationClientProtocol.class);

            if (historyServiceEnabled) {
                historyClient.start();
            }
            if (timelineV2ServiceEnabled) {
                ahsV2Client.start();
            }
        } catch (IOException e) {
            throw new YarnRuntimeException(e);
        }

        // Reinitialize local resource types cache from list of resources pulled
from
        // RM.
        if (loadResourceTypesFromServer) {
            ResourceUtils.reinitializeResources(getResourceTypeInfo());
        }

        super.serviceStart();
    }
```

3.1.1.2 初始化 YANRRunner

```
/**
     * Similar to {@link #YARNRunner(Configuration)} but allowing injecting
     * {@link ResourceMgrDelegate}. Enables mocking and testing.
     *
     * @param conf           the configuration object for the client
     * @param resMgrDelegate the resourcemanager client handle.
     */
    public YARNRunner(Configuration conf, ResourceMgrDelegate resMgrDelegate) {
        // 往下追
        this(
                conf,
                resMgrDelegate,
                // 创建客户端缓存 ClientCache
                new ClientCache(conf, resMgrDelegate));
    }
```

```
/**
     * Similar to {@link YARNRunner#YARNRunner(Configuration, ResourceMgrDelegate)}
     * but allowing injecting {@link ClientCache}. Enable mocking and testing.
     *
     * @param conf           the configuration object
     * @param resMgrDelegate the resource manager delegate
     * @param clientCache    the client cache object.
     */
    public YARNRunner(Configuration conf, ResourceMgrDelegate resMgrDelegate,
                      ClientCache clientCache) {
        this.conf = conf;
        try {
            // ResourceMgrDelegate
            this.resMgrDelegate = resMgrDelegate;
            // ClientCache
            this.clientCache = clientCache;
            // 获取文件系统上下文 FileContext
            this.defaultFileContext = FileContext.getFileContext(this.conf);
        } catch (UnsupportedFileSystemException ufe) {
            throw new RuntimeException("Error in instantiating YarnClient", ufe);
        }
    }
```

3.2 提交任务 Job

```
@InterfaceAudience.Private
@InterfaceStability.Unstable
class JobSubmitter {
  /**
     * Internal method for submitting jobs to the system.
     *
     * <p>The job submission process involves:
     * <ol>
     *   <li>
     *   Checking the input and output specifications of the job.
     *   </li>
     *   <li>
     *   Computing the {@link InputSplit}s for the job.
     *   </li>
     *   <li>
     *   Setup the requisite accounting information for the
     *   {@link DistributedCache} of the job, if necessary.
     *   </li>
     *   <li>
     *   Copying the job's jar and configuration to the map-reduce system
     *   directory on the distributed file-system.
     *   </li>
     *   <li>
     *   Submitting the job to the <code>JobTracker</code> and optionally
     *   monitoring it's status.
     *   </li>
```

```
     * </ol></p>
     *
     * @param job     the configuration to submit
     * @param cluster the handle to the Cluster
     * @throws ClassNotFoundException
     * @throws InterruptedException
     * @throws IOException
     */
    JobStatus submitJobInternal(Job job, Cluster cluster)
            throws ClassNotFoundException, InterruptedException, IOException {

        // validate the jobs output specs
        // 检查任务 Job 输出路径是否存在 如果存在则直接报错
        checkSpecs(job);

        Configuration conf = job.getConfiguration();
        addMRFrameworkToDistributedCache(conf);

        // 获取任务提交临时目录 格式为 /tmp/hadoop-yarn/staging/${user}/.staging
        Path jobStagingArea = JobSubmissionFiles.getStagingDir(cluster, conf);

        // configure the command line options correctly on the submitting dfs
        // 设置当前提交任务节点的主机地址和主机名
        InetAddress ip = InetAddress.getLocalHost();
        if (ip != null) {
            submitHostAddress = ip.getHostAddress();
            submitHostName = ip.getHostName();
            conf.set(MRJobConfig.JOB_SUBMITHOST, submitHostName);
            conf.set(MRJobConfig.JOB_SUBMITHOSTADDR, submitHostAddress);
        }

        // 向 ResourceManager 申请任务 ID
        JobID jobId = submitClient.getNewJobID();
        job.setJobID(jobId);

        // 任务的临时 HDFS 目录 比如 /tmp/hadoop-
yarn/staging/tanbs/.staging/job_1684656010852_0002
        Path submitJobDir = new Path(jobStagingArea, jobId.toString());
        JobStatus status = null;
        try {
            conf.set(MRJobConfig.USER_NAME,
                    UserGroupInformation.getCurrentUser().getShortUserName());
            conf.set("hadoop.http.filter.initializers",

 "org.apache.hadoop.yarn.server.webproxy.amfilter.AmFilterInitializer");
            // 设置任务的临时目录
            conf.set(MRJobConfig.MAPREDUCE_JOB_DIR, submitJobDir.toString());
            LOG.debug("Configuring job " + jobId + " with " + submitJobDir
                    + " as the submit dir");
            // get delegation token for the dir
            TokenCache.obtainTokensForNamenodes(job.getCredentials(),
                    new Path[]{submitJobDir}, conf);
```

```
            populateTokenCache(conf, job.getCredentials());

            // generate a secret to authenticate shuffle transfers
            if (TokenCache.getShuffleSecretKey(job.getCredentials()) == null) {
                KeyGenerator keyGen;
                try {
                    keyGen = KeyGenerator.getInstance(SHUFFLE_KEYGEN_ALGORITHM);
                    keyGen.init(SHUFFLE_KEY_LENGTH);
                } catch (NoSuchAlgorithmException e) {
                    throw new IOException("Error generating shuffle secret key", e);
                }
                SecretKey shuffleKey = keyGen.generateKey();
                TokenCache.setShuffleSecretKey(shuffleKey.getEncoded(),
                        job.getCredentials());
            }
            if (CryptoUtils.isEncryptedSpillEnabled(conf)) {
                conf.setInt(MRJobConfig.MR_AM_MAX_ATTEMPTS, 1);
                LOG.warn("Max job attempts set to 1 since encrypted intermediate" +
                        "data spill is enabled");
            }

            // 提交任务资源到临时目录 (核心提交 job.jar)
            copyAndConfigureFiles(job, submitJobDir);

            // 获取 ${任务提交临时目录}/job.xml 文件目录
            Path submitJobFile = JobSubmissionFiles.getJobConfPath(submitJobDir);

            // Create the splits for the job
            LOG.debug("Creating splits at " + jtFs.makeQualified(submitJobDir));

            // 获取 map 切片信息 (计算切片)
            // 1 创建切片文件 job.split 并上传到 HDFS
            // 2 创建切片元数据文件 job.splitmetainfo 并上传切片元数据信息
            int maps = writeSplits(job, submitJobDir);
            conf.setInt(MRJobConfig.NUM_MAPS, maps);
            // 打印切片个数
            LOG.info("number of splits:" + maps);

            int maxMaps = conf.getInt(MRJobConfig.JOB_MAX_MAP,
                    MRJobConfig.DEFAULT_JOB_MAX_MAP);
            if (maxMaps >= 0 && maxMaps < maps) {
                throw new IllegalArgumentException("The number of map tasks " + maps
+
                        " exceeded limit " + maxMaps);
            }

            // write "queue admins of the queue to which job is being submitted"
            // to job file.
            // 获取任务执行队列 默认 default 队列
            String queue = conf.get(MRJobConfig.QUEUE_NAME,
                    JobConf.DEFAULT_QUEUE_NAME);
```

```
AccessControlList acl = submitClient.getQueueAdmins(queue);
conf.set(toFullPropertyName(queue,
        QueueACL.ADMINISTER_JOBS.getAclName()), acl.getAclString());

// removing jobtoken referrals before copying the jobconf to HDFS
// as the tasks don't need this setting, actually they may break
// because of it if present as the referral will point to a
// different job.
TokenCache.cleanUpTokenReferral(conf);

// 默认 false
if (conf.getBoolean(
        MRJobConfig.JOB_TOKEN_TRACKING_IDS_ENABLED,
        MRJobConfig.DEFAULT_JOB_TOKEN_TRACKING_IDS_ENABLED)) {
    // Add HDFS tracking ids
    ArrayList<String> trackingIds = new ArrayList<String>();
    for (Token<? extends TokenIdentifier> t :
            job.getCredentials().getAllTokens()) {
        trackingIds.add(t.decodeIdentifier().getTrackingId());
    }
    conf.setStrings(MRJobConfig.JOB_TOKEN_TRACKING_IDS,
            trackingIds.toArray(new String[trackingIds.size()]));
}

// Set reservation info if it exists
ReservationId reservationId = job.getReservationId();
if (reservationId != null) {
    conf.set(MRJobConfig.RESERVATION_ID, reservationId.toString());
}

// Write job file to submit dir
// 上传任务配置信息文件 job.xml
writeConf(conf, submitJobFile);

// 到这里才是真正提交任务 上面都是一些环境准备 (比如提交任务的相关信息到 HDFS
// 比如
// 1 任务运行包 job.jar
// 2 切片信息 job.split&job.splitmetainfo
// 3 任务运行配置信息 job.xml
// )
// Now, actually submit the job (using the submit name)
// 打印信息
// 1 Submitting tokens for job: job_1684656010852_0002
// 2 Executing with tokens: []
printTokens(jobId, job.getCredentials());

// 真正客户端提交任务到 Yarn (调用 YARNRunner.submitJob())
status = submitClient.submitJob(
        jobId, // 任务 ID
        submitJobDir.toString(),  // 任务运行相关文件路径目录
        job.getCredentials() // 任务运行凭证
```

```
            );
            if (status != null) {
                return status;
            } else {
                throw new IOException("Could not launch job");
            }
        } finally {
            if (status == null) {
                LOG.info("Cleaning up the staging area " + submitJobDir);
                if (jtFs != null && submitJobDir != null)
                    jtFs.delete(submitJobDir, true);

            }
        }
    }
}
```

3.2.1 向 ResourceManager 的 ClientRMService RPC 申请任务 ApplicationId

```
// 发送 RPC 请求给 ClientRMService 服务返回一个 JobId
// 调用 YARNRunner.getNewJobID()
    @Override
    public JobID getNewJobID() throws IOException, InterruptedException {
        // 向 RM 申请任务 ID
        return resMgrDelegate.getNewJobID();
    }
```

```
    public JobID getNewJobID() throws IOException, InterruptedException {
        try {
            // 发送 RPC 请求给 RM 请求创建一个任务的 ApplicationId 封装成
YarnClientApplication
            this.application = client.createApplication()
                    .getApplicationSubmissionContext();
            this.applicationId = this.application.getApplicationId();
            return TypeConverter.fromYarn(applicationId);
        } catch (YarnException e) {
            throw new IOException(e);
        }
    }
```

3.2.2 获取任务临时目录

```
// 该临时目录是任务提交文件上传路径

// 获取任务提交临时目录 格式为 /tmp/hadoop-yarn/staging/${user}/.staging
Path jobStagingArea = JobSubmissionFiles.getStagingDir(cluster, conf);

// 任务的临时 HDFS 目录 比如 /tmp/hadoop-
yarn/staging/tanbs/.staging/job_1684656010852_0002
Path submitJobDir = new Path(jobStagingArea, jobId.toString());
```

3.2.3 提交 job.jar 信息到临时目录

```
/**
     * configure the jobconf of the user with the command line options of
     * -libjars, -files, -archives.
     *
     * @param job
     * @throws IOException
     */
    private void copyAndConfigureFiles(Job job, Path jobSubmitDir)
            throws IOException {
        Configuration conf = job.getConfiguration();
        // 默认 true
        boolean useWildcards = conf.getBoolean(Job.USE_WILDCARD_FOR_LIBJARS,
                Job.DEFAULT_USE_WILDCARD_FOR_LIBJARS);
        // 创建任务提交资源 JobResourceUploader
        JobResourceUploader rUploader = new JobResourceUploader(jtFs, useWildcards);

        // 提交任务资源到临时目录 (核心提交 job.jar)
        rUploader.uploadResources(job, jobSubmitDir);

        // Get the working directory. If not set, sets it to filesystem working dir
        // This code has been added so that working directory reset before running
        // the job. This is necessary for backward compatibility as other systems
        // might use the public API JobConf#setWorkingDirectory to reset the working
        // directory.
        job.getWorkingDirectory();
    }
```

3.2.4 切片相关信息(job.split, job.splitmetainfo)

```
// 获取 map 切片信息 (计算切片)
// 1 创建切片文件 job.split 并上传到 HDFS
// 2 创建切片元数据文件 job.splitmetainfo 并上传切片元数据信息
int maps = writeSplits(job, submitJobDir);
conf.setInt(MRJobConfig.NUM_MAPS, maps);
```

```
private int writeSplits(org.apache.hadoop.mapreduce.JobContext job,
                            Path jobSubmitDir) throws IOException,
            InterruptedException, ClassNotFoundException {
        JobConf jConf = (JobConf) job.getConfiguration();
        int maps;
        // 一般使用新的 Mapper API
        if (jConf.getUseNewMapper()) {
            // 计算 map 切片个数
            maps = writeNewSplits(job, jobSubmitDir);
        } else {
            maps = writeOldSplits(jConf, jobSubmitDir);
        }
        return maps;
```

```
    }
```

```
@SuppressWarnings("unchecked")
    private <T extends InputSplit>
    int writeNewSplits(JobContext job, Path jobSubmitDir) throws IOException,
            InterruptedException, ClassNotFoundException {
        Configuration conf = job.getConfiguration();
        // 默认 MR 任务输入 TextInputFormat
        InputFormat<?, ?> input =
                ReflectionUtils.newInstance(job.getInputFormatClass(), conf);

        // 计算 map 切片信息 默认调用 TextInputFormat 的父类 FileInputFormat.getSplits()
        List<InputSplit> splits = input.getSplits(job);
        T[] array = (T[]) splits.toArray(new InputSplit[splits.size()]);

        // sort the splits into order based on size, so that the biggest
        // go first
        Arrays.sort(array, new SplitComparator());
        // 将切片信息上传到 HDFS
        // 1 创建切片文件 job.split 并上传到 HDFS
        // 2 创建切片元数据文件 job.splitmetainfo 并上传切片元数据信息
        JobSplitWriter.createSplitFiles(jobSubmitDir, conf,
                jobSubmitDir.getFileSystem(conf), array);
        // 返回切片个数
        return array.length;
    }
```

3.2.4.1 计算切片

```
/**
   * Logically split the set of input files for the job.
   *
   * <p>Each {@link InputSplit} is then assigned to an individual {@link Mapper}
   * for processing.</p>
   *
   * <p><i>Note</i>: The split is a <i>logical</i> split of the inputs and the
   * input files are not physically split into chunks. For e.g. a split could
   * be <i>&lt;input-file-path, start, offset&gt;</i> tuple. The InputFormat
   * also creates the {@link RecordReader} to read the {@link InputSplit}.
   *
   * @param context job configuration.
   * @return an array of {@link InputSplit}s for the job.
   */
  public abstract
    List<InputSplit> getSplits(JobContext context
                               ) throws IOException, InterruptedException;
// 默认调用 TextInputFormat 的父类 FileInputFormat.getSplits()

/**
 * A base class for file-based {@link InputFormat}s.
 *
```

```
 * <p><code>FileInputFormat</code> is the base class for all file-based
 * <code>InputFormat</code>s. This provides a generic implementation of
 * {@link #getSplits(JobContext)}.
 * <p>
 * Implementations of <code>FileInputFormat</code> can also override the
 * {@link #isSplitable(JobContext, Path)} method to prevent input files
 * from being split-up in certain situations. Implementations that may
 * deal with non-splittable files <i>must</i> override this method, since
 * the default implementation assumes splitting is always possible.
 */
@InterfaceAudience.Public
@InterfaceStability.Stable
public abstract class FileInputFormat<K, V> extends InputFormat<K, V> {
    /**
     * Generate the list of files and make them into FileSplits.
     *
     * @param job the job context
     * @throws IOException
     */
    public List<InputSplit> getSplits(JobContext job) throws IOException {
        // 记录切片计算开始时间
        StopWatch sw = new StopWatch().start();

        // 默认 1
        long minSize = Math.max(getFormatMinSplitSize(), getMinSplitSize(job));
        // 默认 Long.MAX_VALUE
        long maxSize = getMaxSplitSize(job);

        // generate splits
        List<InputSplit> splits = new ArrayList<InputSplit>();
        // 获取任务输入的路径 (可能有多个)
        List<FileStatus> files = listStatus(job);

        boolean ignoreDirs = !getInputDirRecursive(job)
                &&
job.getConfiguration().getBoolean(INPUT_DIR_NONRECURSIVE_IGNORE_SUBDIRS, false);

        // 遍历每个任务输入路径
        for (FileStatus file : files) {
            if (ignoreDirs && file.isDirectory()) {
                continue;
            }
            Path path = file.getPath();
            // 获取文件字节长度
            long length = file.getLen();
            if (length != 0) {
                BlockLocation[] blkLocations;

                // 文件存在 HDFS 可能有多个 block
                if (file instanceof LocatedFileStatus) {
                    blkLocations = ((LocatedFileStatus) file).getBlockLocations();
                } else {
```

```
                    FileSystem fs = path.getFileSystem(job.getConfiguration());
                    blkLocations = fs.getFileBlockLocations(file, 0, length);
                }

                // 判断文件是否可以切片
                if (isSplitable(job, path)) {
                    // 获取文件块大小 默认 128MB
                    long blockSize = file.getBlockSize();
                    // 计算一个 map 任务处理多少数据量 ( max(minSize,min(maxSize,
blockSize) )
                    long splitSize = computeSplitSize(blockSize, minSize, maxSize);

                    // 遍历当前文件夹的切片之后残余数据字节数
                    long bytesRemaining = length;
                    // 残余数据字节数 / splitSize(默认 128MB) > 1.1
                    while (((double) bytesRemaining) / splitSize > SPLIT_SLOP) {
                        // 获取文件 block 索引 (第几个 block)
                        int blkIndex = getBlockIndex(blkLocations, length -
bytesRemaining);
                        // 封装切片信息
                        splits.add(makeSplit(
                                path, // 哪个文件路径
                                length - bytesRemaining, // 哪个文件读取数据偏移量
                                splitSize, // block 大小
                                blkLocations[blkIndex].getHosts(), // 哪个 block 存在
在哪些主机上
                                blkLocations[blkIndex].getCachedHosts()
                        ));
                        // 减少切片字节数
                        bytesRemaining -= splitSize;
                    }

                    // 最后是否剩余
                    if (bytesRemaining != 0) {
                        int blkIndex = getBlockIndex(blkLocations, length -
bytesRemaining);
                        splits.add(makeSplit(path, length - bytesRemaining,
bytesRemaining,
                                blkLocations[blkIndex].getHosts(),
                                blkLocations[blkIndex].getCachedHosts()));
                    }
                } else { // not splitable
                    if (LOG.isDebugEnabled()) {
                        // Log only if the file is big enough to be splitted
                        if (length > Math.min(file.getBlockSize(), minSize)) {
                            LOG.debug("File is not splittable so no parallelization
"
                                    + "is possible: " + file.getPath());
                        }
                    }
                    splits.add(makeSplit(path, 0, length,
blkLocations[0].getHosts(),
```

```
                            blkLocations[0].getCachedHosts()));
                }
            } else {
                //Create empty hosts array for zero length files
                splits.add(makeSplit(path, 0, length, new String[0]));
            }
        }
        // Save the number of input files for metrics/loadgen
        job.getConfiguration().setLong(NUM_INPUT_FILES, files.size());
        sw.stop();
        if (LOG.isDebugEnabled()) {
            LOG.debug("Total # of splits generated by getSplits: " + splits.size()
                    + ", TimeTaken: " + sw.now(TimeUnit.MILLISECONDS));
        }
        return splits;
    }
}
```

3.2.4.2 上传切片信息到临时目录

```
// 1 创建切片文件 job.split 并上传到 HDFS
// 2 创建切片元数据文件 job.splitmetainfo 并上传切片元数据信息

/**
 * The class that is used by the Job clients to write splits (both the meta
 * and the raw bytes parts)
 */
@InterfaceAudience.Private
@InterfaceStability.Unstable
public class JobSplitWriter {
 public static <T extends InputSplit> void createSplitFiles(Path jobSubmitDir,
                                                             Configuration conf,
FileSystem fs, T[] splits)
            throws IOException, InterruptedException {
        // 创建切片文件 job.split 并上传到 HDFS
        FSDataOutputStream out = createFile(fs,
                JobSubmissionFiles.getJobSplitFile(jobSubmitDir), conf);
        SplitMetaInfo[] info = writeNewSplits(conf, splits, out);
        out.close();

        // 创建切片元数据文件 job.splitmetainfo 并上传切片元数据信息
        writeJobSplitMetaInfo(fs,
JobSubmissionFiles.getJobSplitMetaFile(jobSubmitDir),
                new FsPermission(JobSubmissionFiles.JOB_FILE_PERMISSION),
splitVersion,
                info);
    }
}
```

3.2.5 上传任务配置信息文件 job.xml

```
// 获取 ${任务提交临时目录}/job.xml 文件目录
Path submitJobFile = JobSubmissionFiles.getJobConfPath(submitJobDir);
......
// Write job file to submit dir
// 上传任务配置信息文件 job.xml
writeConf(conf, submitJobFile);
```

```
    private void writeConf(Configuration conf, Path jobFile)
            throws IOException {
        // Write job file to JobTracker's fs
        FSDataOutputStream out =
                FileSystem.create(jtFs, jobFile,
                        new FsPermission(JobSubmissionFiles.JOB_FILE_PERMISSION));
        try {
            // 写任务配置信息 job.xml
            conf.writeXml(out);
        } finally {
            out.close();
        }
    }
```

3.2.6 RPC 客户端提交任务到 Yarn (调用 YARNRunner.submitJob())

```
// RPC 客户端提交任务到 Yarn (调用 YARNRunner.submitJob())
            status = submitClient.submitJob(
                    jobId, // 任务 ID
                    submitJobDir.toString(),  // 任务运行相关文件路径目录
                    job.getCredentials() // 任务运行凭证
            );
```

```
@Override
    public JobStatus submitJob(JobID jobId, String jobSubmitDir, Credentials ts)
            throws IOException, InterruptedException {

        addHistoryToken(ts);

        // 将提交任务信息 (任务配置文件路径) 封装成 ApplicationSubmissionContext
        ApplicationSubmissionContext appContext =
                createApplicationSubmissionContext(conf, jobSubmitDir, ts);

        // Submit to ResourceManager
        try {
            // 提交任务 (发送 RPC 请求)
            ApplicationId applicationId =
                    resMgrDelegate.submitApplication(appContext);

            // 获取任务执行信息
```

```
            ApplicationReport appMaster = resMgrDelegate
                    .getApplicationReport(applicationId);
            String diagnostics =
                    (appMaster == null ?
                            "application report is null" :
appMaster.getDiagnostics());
            if (appMaster == null
                    || appMaster.getYarnApplicationState() ==
YarnApplicationState.FAILED
                    || appMaster.getYarnApplicationState() ==
YarnApplicationState.KILLED) {
                throw new IOException("Failed to run job : " +
                        diagnostics);
            }
            return clientCache.getClient(jobId).getJobStatus(jobId);
        } catch (YarnException e) {
            throw new IOException(e);
        }
    }
```

3.2.6.1 提交任务信息封装成 ApplicationSubmissionContext (启动 MRAppMaster)

```
/**
     * Constructs all the necessary information to start the MR AM.
     *
     * @param jobConf      the configuration for the MR job
     * @param jobSubmitDir the directory path for the job
     * @param ts           the security credentials for the job
     * @return ApplicationSubmissionContext
     * @throws IOException on IO error (e.g. path resolution)
     */
    public ApplicationSubmissionContext createApplicationSubmissionContext(
            Configuration jobConf, String jobSubmitDir, Credentials ts)
            throws IOException {
        // 获取任务 ApplicationId
        ApplicationId applicationId = resMgrDelegate.getApplicationId();

        // Setup LocalResources
        // 设置本地资源配置到任务
        // 1 job.xml -> LocalResource
        // 2 job.jar -> LocalResource
        // 3 jobSubmitDir/job.split -> LocalResource
        // 4 jobSubmitDir/job.splitmetainfo -> LocalResource
        Map<String, LocalResource> localResources =
                setupLocalResources(jobConf, jobSubmitDir);

        // Setup security tokens
        DataOutputBuffer dob = new DataOutputBuffer();
        ts.writeTokenStorageToStream(dob);
        ByteBuffer securityTokens =
                ByteBuffer.wrap(dob.getData(), 0, dob.getLength());
```

```
        // Setup ContainerLaunchContext for AM container
        // 构建启动 AM 命令
        /**
         * 基本启动 AM 命令模版
         * ${JAVA_HOME}/bin/java
         * -Djava.io.tmpdir=./tmp
         * -Dlog4j.configuration=container-log4j.properties
         * -Dyarn.app.container.log.dir=<LOG_DIR>
         * -Dyarn.app.container.log.filesize=0
         * -Dhadoop.root.logger=INFO,CLA
         * -Dhadoop.root.logfile=syslog
         * -Xmx1024m
         * org.apache.hadoop.mapreduce.v2.app.MRAppMaster
         * 1><LOG_DIR>.stdout
         * 2><LOG_DIR>.stderr
         */
        List<String> vargs = setupAMCommand(jobConf);
        // 创建启动 AM 容器上下文 (也即拼接最终启动 AM 容器命令以及启动环境变量)
        ContainerLaunchContext amContainer = setupContainerLaunchContextForAM(
                jobConf, localResources, securityTokens, vargs);

        String regex = conf.get(MRJobConfig.MR_JOB_SEND_TOKEN_CONF);
        if (regex != null && !regex.isEmpty()) {
            setTokenRenewerConf(amContainer, conf, regex);
        }

        Collection<String> tagsFromConf =
                jobConf.getTrimmedStringCollection(MRJobConfig.JOB_TAGS);

        // Set up the ApplicationSubmissionContext
        // 创建任务提交上下文对象 ApplicationSubmissionContext
        ApplicationSubmissionContext appContext =
                recordFactory.newRecordInstance(ApplicationSubmissionContext.class);
        // 任务提交上下文对象设置任务应用 ID
        appContext.setApplicationId(applicationId);                // ApplicationId
        // 设置队列 默认 default
        appContext.setQueue(                                       // Queue name
                jobConf.get(JobContext.QUEUE_NAME,
                        YarnConfiguration.DEFAULT_QUEUE_NAME));

        // add reservationID if present
        // 默认 null
        ReservationId reservationID = null;
        try {
            reservationID =
                    ReservationId.parseReservationId(jobConf
                            .get(JobContext.RESERVATION_ID));
        } catch (NumberFormatException e) {
            // throw exception as reservationid as is invalid
            String errMsg =
                    "Invalid reservationId: " +
jobConf.get(JobContext.RESERVATION_ID)
```

```
                        + " specified for the app: " + applicationId;
            LOG.warn(errMsg);
            throw new IOException(errMsg);
        }
        if (reservationID != null) {
            appContext.setReservationID(reservationID);
            LOG.info("SUBMITTING ApplicationSubmissionContext app:" + applicationId
                    + " to queue:" + appContext.getQueue() + " with reservationId:"
                    + appContext.getReservationID());
        }

        // 设置任务名称 (Job.getInstance(conf, "word count") word count)
        appContext.setApplicationName(                              // Job name
                jobConf.get(JobContext.JOB_NAME,
                        YarnConfiguration.DEFAULT_APPLICATION_NAME));
        appContext.setCancelTokensWhenComplete(
                conf.getBoolean(MRJobConfig.JOB_CANCEL_DELEGATION_TOKEN, true));
        // 设置 AM 启动容器
        appContext.setAMContainerSpec(amContainer);         // AM Container
        // 设置最大重启次数 默认 2
        appContext.setMaxAppAttempts(
                conf.getInt(MRJobConfig.MR_AM_MAX_ATTEMPTS,
                        MRJobConfig.DEFAULT_MR_AM_MAX_ATTEMPTS));

        // Setup the AM ResourceRequests
        // 生成启动 AM 容器资源请求 (默认内存 1536 MB 1 CPU *)
        List<ResourceRequest> amResourceRequests = generateResourceRequests();
        appContext.setAMContainerResourceRequests(amResourceRequests);

        // set labels for the AM container requests if present
        String amNodelabelExpression = conf.get(MRJobConfig.AM_NODE_LABEL_EXP);
        if (null != amNodelabelExpression
                && amNodelabelExpression.trim().length() != 0) {
            for (ResourceRequest amResourceRequest : amResourceRequests) {

amResourceRequest.setNodeLabelExpression(amNodelabelExpression.trim());
            }
        }
        // set labels for the Job containers
        appContext.setNodeLabelExpression(jobConf
                .get(JobContext.JOB_NODE_LABEL_EXP));

        // 设置任务类型为 MAPREDUCE
        appContext.setApplicationType(MRJobConfig.MR_APPLICATION_TYPE);
        if (tagsFromConf != null && !tagsFromConf.isEmpty()) {
            appContext.setApplicationTags(new HashSet<>(tagsFromConf));
        }

        String jobPriority = jobConf.get(MRJobConfig.PRIORITY);
        if (jobPriority != null) {
            int iPriority;
            try {
```

```
                iPriority = TypeConverter.toYarnApplicationPriority(jobPriority);
            } catch (IllegalArgumentException e) {
                iPriority = Integer.parseInt(jobPriority);
            }
            appContext.setPriority(Priority.newInstance(iPriority));
        }

        // 返回 ApplicationSubmissionContext
        return appContext;
    }
```

3.2.6.2 提交任务(调用 YarnClientImpl.submitApplication())

```
@Override
    public ApplicationId
    submitApplication(ApplicationSubmissionContext appContext)
            throws YarnException, IOException {
        // 提交任务到 YARN (调用 YarnClientImpl.submitApplication())
        return client.submitApplication(appContext);
    }
```

```
@Override
    public ApplicationId
    submitApplication(ApplicationSubmissionContext appContext)
            throws YarnException, IOException {
        // 判断提交任务的 ApplicationId 是否为空
        ApplicationId applicationId = appContext.getApplicationId();
        if (applicationId == null) {
            throw new ApplicationIdNotProvidedException(
                    "ApplicationId is not provided in
ApplicationSubmissionContext");
        }
        // 封装任务提交上下文对象为 RPC 请求对象
        SubmitApplicationRequest request =
                Records.newRecord(SubmitApplicationRequest.class);
        request.setApplicationSubmissionContext(appContext);

        // Automatically add the timeline DT into the CLC
        // Only when the security and the timeline service are both enabled
        if (isSecurityEnabled() && timelineV1ServiceEnabled) {
            addTimelineDelegationToken(appContext.getAMContainerSpec());
        }

        //TODO: YARN-1763:Handle RM failovers during the submitApplication call.
        // 提交任务 (调用 ClientRMService.submitApplication())
        rmClient.submitApplication(request);

        int pollCount = 0;
        long startTime = System.currentTimeMillis();
        EnumSet<YarnApplicationState> waitingStates =
                EnumSet.of(YarnApplicationState.NEW,
```

```
                        YarnApplicationState.NEW_SAVING,
                        YarnApplicationState.SUBMITTED);
        EnumSet<YarnApplicationState> failToSubmitStates =
                EnumSet.of(YarnApplicationState.FAILED,
                        YarnApplicationState.KILLED);
        while (true) {
            try {
                // 持续循环判断提交的任务是否成功
                ApplicationReport appReport = getApplicationReport(applicationId);
                YarnApplicationState state = appReport.getYarnApplicationState();
                if (!waitingStates.contains(state)) {
                    if (failToSubmitStates.contains(state)) {
                        throw new YarnException("Failed to submit " + applicationId
+
                                " to YARN : " + appReport.getDiagnostics());
                    }
                    // Submitted application application_1684656010852_0002
                    LOG.info("Submitted application " + applicationId);
                    break;
                }

                long elapsedMillis = System.currentTimeMillis() - startTime;
                if (enforceAsyncAPITimeout() &&
                        elapsedMillis >= asyncApiPollTimeoutMillis) {
                    throw new YarnException("Timed out while waiting for application
" +
                            applicationId + " to be submitted successfully");
                }

                // Notify the client through the log every 10 poll, in case the
client
                // is blocked here too long.
                if (++pollCount % 10 == 0) {
                    LOG.info("Application submission is not finished, " +
                            "submitted application " + applicationId +
                            " is still in " + state);
                }
                try {
                    Thread.sleep(submitPollIntervalMillis);
                } catch (InterruptedException ie) {
                    String msg = "Interrupted while waiting for application "
                            + applicationId + " to be successfully submitted.";
                    LOG.error(msg);
                    throw new YarnException(msg, ie);
                }
            } catch (ApplicationNotFoundException ex) {
                // FailOver or RM restart happens before RMStateStore saves
                // ApplicationState
                LOG.info("Re-submit application " + applicationId + "with the " +
                        "same ApplicationSubmissionContext");
                rmClient.submitApplication(request);
            }
```

```
        }

        return applicationId;
    }
```

```
@Override
    public ApplicationId
    submitApplication(ApplicationSubmissionContext appContext)
            throws YarnException, IOException {
        // 判断提交任务的 ApplicationId 是否为空
        ApplicationId applicationId = appContext.getApplicationId();
        if (applicationId == null) {
            throw new ApplicationIdNotProvidedException(
                    "ApplicationId is not provided in
ApplicationSubmissionContext");
        }
        // 封装任务提交上下文对象为 RPC 请求对象
        SubmitApplicationRequest request =
                Records.newRecord(SubmitApplicationRequest.class);
        request.setApplicationSubmissionContext(appContext);

        // Automatically add the timeline DT into the CLC
        // Only when the security and the timeline service are both enabled
        if (isSecurityEnabled() && timelineV1ServiceEnabled) {
            addTimelineDelegationToken(appContext.getAMContainerSpec());
        }

        //TODO: YARN-1763:Handle RM failovers during the submitApplication call.
        // 提交任务 (调用 ClientRMService.submitApplication())
        rmClient.submitApplication(request);

        int pollCount = 0;
        long startTime = System.currentTimeMillis();
        EnumSet<YarnApplicationState> waitingStates =
                EnumSet.of(YarnApplicationState.NEW,
                        YarnApplicationState.NEW_SAVING,
                        YarnApplicationState.SUBMITTED);
        EnumSet<YarnApplicationState> failToSubmitStates =
                EnumSet.of(YarnApplicationState.FAILED,
                        YarnApplicationState.KILLED);
        while (true) {
            try {
                // 持续循环判断提交的任务是否成功
                ApplicationReport appReport = getApplicationReport(applicationId);
                YarnApplicationState state = appReport.getYarnApplicationState();
                if (!waitingStates.contains(state)) {
                    if (failToSubmitStates.contains(state)) {
                        throw new YarnException("Failed to submit " + applicationId
+
                                " to YARN : " + appReport.getDiagnostics());
                    }
```

```
                    // Submitted application application_1684656010852_0002
                    LOG.info("Submitted application " + applicationId);
                    break;
                }

                long elapsedMillis = System.currentTimeMillis() - startTime;
                if (enforceAsyncAPITimeout() &&
                        elapsedMillis >= asyncApiPollTimeoutMillis) {
                    throw new YarnException("Timed out while waiting for application
" +
                            applicationId + " to be submitted successfully");
                }

                // Notify the client through the log every 10 poll, in case the
client
                // is blocked here too long.
                if (++pollCount % 10 == 0) {
                    LOG.info("Application submission is not finished, " +
                            "submitted application " + applicationId +
                            " is still in " + state);
                }
                try {
                    Thread.sleep(submitPollIntervalMillis);
                } catch (InterruptedException ie) {
                    String msg = "Interrupted while waiting for application "
                            + applicationId + " to be successfully submitted.";
                    LOG.error(msg);
                    throw new YarnException(msg, ie);
                }
            } catch (ApplicationNotFoundException ex) {
                // FailOver or RM restart happens before RMStateStore saves
                // ApplicationState
                LOG.info("Re-submit application " + applicationId + "with the " +
                        "same ApplicationSubmissionContext");
                rmClient.submitApplication(request);
            }
        }

        return applicationId;
    }
```

四 ResourceManager 的 ClientRMService 接收到客户端发送的 RPC 请求 (启动 AM)

```
// 提交任务 (调用 ClientRMService.submitApplication())
@Override
    public SubmitApplicationResponse submitApplication(
            SubmitApplicationRequest request) throws YarnException, IOException {
        // RPC 请求信息
        ApplicationSubmissionContext submissionContext = request
```

```
                .getApplicationSubmissionContext();
        ApplicationId applicationId = submissionContext.getApplicationId();
        CallerContext callerContext = CallerContext.getCurrent();

        // ApplicationSubmissionContext needs to be validated for safety - only
        // those fields that are independent of the RM's configuration will be
        // checked here, those that are dependent on RM configuration are validated
        // in RMAppManager.

        String user = null;
        try {
            // Safety
            user = UserGroupInformation.getCurrentUser().getShortUserName();
        } catch (IOException ie) {
            LOG.warn("Unable to get the current user.", ie);
            RMAuditLogger.logFailure(user, AuditConstants.SUBMIT_APP_REQUEST,
                    ie.getMessage(), "ClientRMService",
                    "Exception in submitting application", applicationId,
callerContext,
                    submissionContext.getQueue());
            throw RPCUtil.getRemoteException(ie);
        }

        if (timelineServiceV2Enabled) {
            // Sanity check for flow run
            String value = null;
            try {
                for (String tag : submissionContext.getApplicationTags()) {
                    if (tag.startsWith(TimelineUtils.FLOW_RUN_ID_TAG_PREFIX + ":")
||
                            tag.startsWith(

 TimelineUtils.FLOW_RUN_ID_TAG_PREFIX.toLowerCase() + ":")) {
                        value =
tag.substring(TimelineUtils.FLOW_RUN_ID_TAG_PREFIX.length()
                                + 1);
                        // In order to check the number format
                        Long.valueOf(value);
                    }
                }
            } catch (NumberFormatException e) {
                LOG.warn("Invalid to flow run: " + value +
                        ". Flow run should be a long integer", e);
                RMAuditLogger.logFailure(user, AuditConstants.SUBMIT_APP_REQUEST,
                        e.getMessage(), "ClientRMService",
                        "Exception in submitting application", applicationId,
                        submissionContext.getQueue());
                throw RPCUtil.getRemoteException(e);
            }
        }

        // Check whether app has already been put into rmContext,
```

```
        // If it is, simply return the response
        if (rmContext.getRMApps().get(applicationId) != null) {
            LOG.info("This is an earlier submitted application: " + applicationId);
            return SubmitApplicationResponse.newInstance();
        }

        ByteBuffer tokenConf =
                submissionContext.getAMContainerSpec()
                        .getTokensConf();
        if (tokenConf != null) {
            int maxSize = getConfig()
                    .getInt(YarnConfiguration.RM_DELEGATION_TOKEN_MAX_CONF_SIZE,

 YarnConfiguration.DEFAULT_RM_DELEGATION_TOKEN_MAX_CONF_SIZE_BYTES);
            LOG.info("Using app provided configurations for delegation token
renewal,"
                    + " total size = " + tokenConf.capacity());
            if (tokenConf.capacity() > maxSize) {
                throw new YarnException(
                        "Exceed " +
YarnConfiguration.RM_DELEGATION_TOKEN_MAX_CONF_SIZE
                                + " = " + maxSize + " bytes, current conf size = "
                                + tokenConf.capacity() + " bytes.");
            }
        }

        if (submissionContext.getQueue() == null) {
            submissionContext.setQueue(YarnConfiguration.DEFAULT_QUEUE_NAME);
        }
        if (submissionContext.getApplicationName() == null) {
            submissionContext.setApplicationName(
                    YarnConfiguration.DEFAULT_APPLICATION_NAME);
        }

        if (submissionContext.getApplicationType() == null) {
            submissionContext
                    .setApplicationType(YarnConfiguration.DEFAULT_APPLICATION_TYPE);
        } else {
            if (submissionContext.getApplicationType().length() >
YarnConfiguration.APPLICATION_TYPE_LENGTH) {
                submissionContext.setApplicationType(submissionContext
                        .getApplicationType().substring(0,
                                YarnConfiguration.APPLICATION_TYPE_LENGTH));
            }
        }

        ReservationId reservationId = request.getApplicationSubmissionContext()
                .getReservationID();
        checkReservationACLs(submissionContext.getQueue(), AuditConstants
                .SUBMIT_RESERVATION_REQUEST, reservationId);

        try {
```

```
            // call RMAppManager to submit application directly
            // 直接提交任务到 RMAppManager
            rmAppManager.submitApplication(
                    submissionContext,
                    System.currentTimeMillis(),
                    user);

            // Application with id 5 submitted by user tanbs
            LOG.info("Application with id " + applicationId.getId() +
                    " submitted by user " + user);
            RMAuditLogger.logSuccess(user, AuditConstants.SUBMIT_APP_REQUEST,
                    "ClientRMService", applicationId, callerContext,
                    submissionContext.getQueue());
        } catch (YarnException e) {
            LOG.info("Exception in submitting " + applicationId, e);
            RMAuditLogger.logFailure(user, AuditConstants.SUBMIT_APP_REQUEST,
                    e.getMessage(), "ClientRMService",
                    "Exception in submitting application", applicationId,
callerContext,
                    submissionContext.getQueue());
            throw e;
        }

        return recordFactory
                .newRecordInstance(SubmitApplicationResponse.class);
    }
```

4.1 备注说明

```
// 由于提交启动 MRAppMaster RPC 请求涉及很多的状态机转换 故文档不暂时 需要的话请阅读源码进行理解
// 这里直接来到准备启动 MRAppMaster 代码 (注意的是 此时启动 AM 已经被分配了容器 Container)
```

4.2 启动 AM (MRAppMaster) 调用 ApplicationMasterLauncher.handle()

```
public class ApplicationMasterLauncher extends AbstractService implements
        EventHandler<AMLauncherEvent> {
 @Override
    public synchronized void handle(AMLauncherEvent appEvent) {

        // event = AMLauncherEvent
        // evenType = AMLauncherEventType.LAUNCH

        AMLauncherEventType event = appEvent.getType();
        RMAppAttempt application = appEvent.getAppAttempt();
        switch (event) {
            case LAUNCH:
                // 启动 AM
                launch(application);
                break;
```

```
            case CLEANUP:
                cleanup(application);
                break;
            default:
                break;
        }
    }
}
```

```
    private void launch(RMAppAttempt application) {

        // 创建 AMLauncher
        Runnable launcher = createRunnableLauncher(application,
                AMLauncherEventType.LAUNCH);
        // 将启动 AM 的 AMLauncher 任务存放到阻塞队列等待 LauncherThread 线程处理
        // 调用 LauncherThread.run()
        masterEvents.add(launcher);
    }
```

```
 private class LauncherThread extends Thread {

        public LauncherThread() {
            super("ApplicationMaster Launcher");
        }

        @Override
        public void run() {
            while (!this.isInterrupted()) {
                Runnable toLaunch;
                try {
                    // 拉取需要启动 ApplicationMaster
                    toLaunch = masterEvents.take();
                    // 执行启动 AM 调用 AMLauncher.run()
                    launcherPool.execute(toLaunch);
                } catch (InterruptedException e) {
                    LOG.warn(this.getClass().getName() + " interrupted.
Returning.");
                    return;
                }
            }
        }
    }
```

4.2.1 执行启动 AM 调用 AMLauncher.run()

```
/**
 * The launch of the AM itself.
 */
public class AMLauncher implements Runnable {
     @SuppressWarnings("unchecked")
```

```
    public void run() {
        switch (eventType) {
            // 启动 AM 事件
            case LAUNCH:
                try {
                    // Launching masterappattempt_1684656010852_0005_000001
                    LOG.info("Launching master" + application.getAppAttemptId());
                    // 启动 AM
                    launch();
                    // 调用
RMAppAttemptEventDispatcher.handle(RMAppAttemptEventType.LAUNCHED)
                    handler.handle(new
RMAppAttemptEvent(application.getAppAttemptId(),
                            RMAppAttemptEventType.LAUNCHED,
System.currentTimeMillis()));
                } catch (Exception ie) {
                    onAMLaunchFailed(masterContainer.getId(), ie);
                }
                break;
            case CLEANUP:
                try {
                    LOG.info("Cleaning master " + application.getAppAttemptId());
                    cleanup();
                } catch (IOException ie) {
                    LOG.info("Error cleaning master ", ie);
                } catch (YarnException e) {
                    StringBuilder sb = new StringBuilder("Container ");
                    sb.append(masterContainer.getId().toString());
                    sb.append(" is not handled by this NodeManager");
                    if (!e.getMessage().contains(sb.toString())) {
                        // Ignoring if container is already killed by Node Manager.
                        LOG.info("Error cleaning master ", e);
                    }
                }
                break;
            default:
                LOG.warn("Received unknown event-type " + eventType + ".
Ignoring.");
                break;
        }
    }
}
```

```
private void launch() throws IOException, YarnException {
        // 根据 AM 分配的 NodeId( NodeManager 的 RPC 服务 ContainerManagerImpl)
        // 获取 ContainerManagerImpl 的 RPC 客户端代理
        // 通讯协议接口为 ContainerManagementProtocol
        connect();
        // 获取启动 AM 的容器信息
        ContainerId masterContainerID = masterContainer.getId();
        // 获取启动 AM 的上下文信息
```

```
        ApplicationSubmissionContext applicationContext =
                application.getSubmissionContext();
        //  Setting up container Container:
        //  [ContainerId: container_1684656010852_0005_01_000001,
        //  AllocationRequestId: -1,
        //  Version: 0,
        //  NodeId: hadoop104:35189,
        //  NodeHttpAddress: hadoop104:8042,
        //  Resource: <memory:1536, vCores:1>,
        //  Priority: 0,
        //  Token: Token { kind: ContainerToken, service: 192.168.6.104:35189 },
        //  ExecutionType: GUARANTEED, ]
        //  for AM appattempt_1684656010852_0005_000001
        LOG.info("Setting up container " + masterContainer
                + " for AM " + application.getAppAttemptId());

        // 创建启动容器的上下文 ContainerLaunchContext
        // (也即封装启动 AM 的上下文、容器信息、其他 Token、以及环境变量相关等等)
        ContainerLaunchContext launchContext =
                createAMContainerLaunchContext(applicationContext,
masterContainerID);

        // 创建启动容器请求体 (封装启动容器的上下文 ContainerLaunchContext )
        StartContainerRequest scRequest =
                StartContainerRequest.newInstance(launchContext,
                        masterContainer.getContainerToken());
        List<StartContainerRequest> list = new ArrayList<StartContainerRequest>();
        list.add(scRequest);
        StartContainersRequest allRequests =
                StartContainersRequest.newInstance(list);

        // 发送 RPC 请求到 NodeManager 的 ContainerManagerImpl RPC 服务
        // 调用 ContainerManagerImpl.startContainers()
        StartContainersResponse response =
                containerMgrProxy.startContainers(allRequests);
        if (response.getFailedRequests() != null
                && response.getFailedRequests().containsKey(masterContainerID)) {
            Throwable t =

 response.getFailedRequests().get(masterContainerID).deSerialize();
            parseAndThrowException(t);
        } else {
            // Done launching container Container:
            // [ContainerId: container_1684656010852_0005_01_000001,
            // AllocationRequestId: -1,
            // Version: 0,
            // NodeId: hadoop104:35189,
            // NodeHttpAddress: hadoop104:8042,
            // Resource: <memory:1536, vCores:1>,
            // Priority: 0,
            // Token: Token { kind: ContainerToken, service: 192.168.6.104:35189 },
            // ExecutionType: GUARANTEED, ]
```

```
            // for AM appattempt_1684656010852_0005_000001
            LOG.info("Done launching container " + masterContainer + " for AM "
                    + application.getAppAttemptId());
        }
    }
```

4.2.1.1 获取 ContainerManagerImpl 的 RPC 客户端代理

```
    private void connect() throws IOException {
        // 获取启动 AM 的容器信息 (也即 NodeManager 信息)
        ContainerId masterContainerID = masterContainer.getId();
        // 创建 NodeManager 服务的 ContainerManagerImpl RPC Server 的客户端代理
        // 通讯协议接口为 ContainerManagementProtocol
        containerMgrProxy = getContainerMgrProxy(masterContainerID);
    }
```

4.2.1.2 发送 RPC 请求到 NodeManager 的 ContainerManagerImpl RPC 服务启动容器

```
// 调用 ContainerManagerImpl.startContainers()
/**
     * Start a list of containers on this NodeManager.
     */
    @Override
    public StartContainersResponse startContainers(
            StartContainersRequest requests) throws YarnException, IOException {
        // 获取 RPC 客户端用户组信息并执行认证相关工作
        UserGroupInformation remoteUgi = getRemoteUgi();
        NMTokenIdentifier nmTokenIdentifier = selectNMTokenIdentifier(remoteUgi);
        authorizeUser(remoteUgi, nmTokenIdentifier);

        List<ContainerId> succeededContainers = new ArrayList<ContainerId>();
        Map<ContainerId, SerializedException> failedContainers =
                new HashMap<ContainerId, SerializedException>();
        // Synchronize with NodeStatusUpdaterImpl#registerWithRM
        // to avoid race condition during NM-RM resync (due to RM restart) while a
        // container is being started, in particular when the container has not yet
        // been added to the containers map in NMContext.
        synchronized (this.context) {
            // 针对启动 AM 而言 StartContainerRequest 只有一个
            for (StartContainerRequest request : requests
                    .getStartContainerRequests()) {
                ContainerId containerId = null;
                try {
                    if (request.getContainerToken() == null
                            || request.getContainerToken().getIdentifier() == null)
{
                        throw new IOException(INVALID_CONTAINERTOKEN_MSG);
                    }

                    ContainerTokenIdentifier containerTokenIdentifier = BuilderUtils
```

```
.newContainerTokenIdentifier(request.getContainerToken());

 verifyAndGetContainerTokenIdentifier(request.getContainerToken(),
                            containerTokenIdentifier);

                    // 获取容器信息
                    containerId = containerTokenIdentifier.getContainerID();

                    // Initialize the AMRMProxy service instance only if the
container is of
                    // type AM and if the AMRMProxy service is enabled
                    if (amrmProxyEnabled &&
containerTokenIdentifier.getContainerType()
                            .equals(ContainerType.APPLICATION_MASTER)) {

 this.getAMRMProxyService().processApplicationStartRequest(request);
                    }
                    performContainerPreStartChecks(nmTokenIdentifier, request,
                            containerTokenIdentifier);

                    // 启动容器
                    startContainerInternal(containerTokenIdentifier, request);
                    // 表示启动容器成功
                    succeededContainers.add(containerId);
                } catch (YarnException e) {
                    failedContainers.put(containerId,
SerializedException.newInstance(e));
                } catch (InvalidToken ie) {
                    failedContainers
                            .put(containerId, SerializedException.newInstance(ie));
                    throw ie;
                } catch (IOException e) {
                    throw RPCUtil.getRemoteException(e);
                }
            }
            // 返回 RPC 请求响应
            return StartContainersResponse
                    .newInstance(getAuxServiceMetaData(), succeededContainers,
                            failedContainers);
        }
    }
```

```
@SuppressWarnings("unchecked")
    protected void startContainerInternal(
            ContainerTokenIdentifier containerTokenIdentifier,
            StartContainerRequest request) throws YarnException, IOException {

        ContainerId containerId = containerTokenIdentifier.getContainerID();
        String containerIdStr = containerId.toString();
        String user = containerTokenIdentifier.getApplicationSubmitter();
```

```
        // start request for container_1684656010852_0005_01_000001 by user tanbs
        LOG.info("Start request for " + containerIdStr + " by user " + user);

        // 获取启动容器上下文
        ContainerLaunchContext launchContext = request.getContainerLaunchContext();

        // Sanity check for local resources
        // 对本地资源进行完整性检查
        for (Map.Entry<String, LocalResource> rsrc : launchContext
                .getLocalResources().entrySet()) {
            if (rsrc.getValue() == null || rsrc.getValue().getResource() == null) {
                throw new YarnException(
                        "Null resource URL for local resource " + rsrc.getKey() + "
: " + rsrc.getValue());
            } else if (rsrc.getValue().getType() == null) {
                throw new YarnException(
                        "Null resource type for local resource " + rsrc.getKey() + "
: " + rsrc.getValue());
            } else if (rsrc.getValue().getVisibility() == null) {
                throw new YarnException(
                        "Null resource visibility for local resource " +
rsrc.getKey() + " : " + rsrc.getValue());
            }
        }

        Credentials credentials =
                YarnServerSecurityUtils.parseCredentials(launchContext);

        long containerStartTime = SystemClock.getInstance().getTime();
        // 创建 ContainerImpl (再次封装容器)
        Container container =
                new ContainerImpl(
                        getConfig(),
                        this.dispatcher,
                        launchContext,
                        credentials,
                        metrics,
                        containerTokenIdentifier,
                        context,
                        containerStartTime);
        // 获取启动容器的任务 ApplicationId
        ApplicationId applicationID =
                containerId.getApplicationAttemptId().getApplicationId();
        // NodeManager 的上下文对象缓存容器 (ContainerId -> ContainerImpl)
        if (context.getContainers().putIfAbsent(containerId, container) != null) {
            NMAuditLogger.logFailure(user, AuditConstants.START_CONTAINER,
                    "ContainerManagerImpl", "Container already running on this
node!",
                    applicationID, containerId);
            throw RPCUtil.getRemoteException("Container " + containerIdStr
                    + " already is running on this node!!");
```

```
        }

        this.readLock.lock();
        try {
            if (!isServiceStopped()) {
                // 判断当前 NodeManager 是否已经启动过 ApplicationId 对应的任务
                // 对于启动 AM 而言 还没有启动 ApplicationId 对应的任务
                if (!context.getApplications().containsKey(applicationID)) {
                    // Create the application
                    // populate the flow context from the launch context if the
timeline
                    // service v.2 is enabled
                    FlowContext flowContext =
                            getFlowContext(launchContext, applicationID);

                    // 创建 ApplicationImpl
                    Application application =
                            new ApplicationImpl(dispatcher, user, flowContext,
                                    applicationID, credentials, context);
                    if (context.getApplications().putIfAbsent(applicationID,
                            application) == null) {
                        // Creating a new application reference for app
                        // application_1684656010852_0005
                        LOG.info("Creating a new application reference for app "
                                + applicationID);
                        LogAggregationContext logAggregationContext =
                                containerTokenIdentifier.getLogAggregationContext();
                        Map<ApplicationAccessType, String> appAcls =
                                container.getLaunchContext().getApplicationACLs();
                        // 存储 ApplicationId (ApplicationId ->
ContainerManagerApplicationProto)
                        context.getNMStateStore().storeApplication(applicationID,
                                buildAppProto(applicationID, user, credentials,
appAcls,
                                        logAggregationContext, flowContext));
                        // 调用
ApplicationEventDispatcher.handle(ApplicationEventType.INIT_APPLICATION)
                        // .....
                        // 最终调用 ContainerScheduler.handle(SCHEDULE_CONTAINER) 将启
动容器放入缓存队列
                        // 判断是否启动容器 (强制启动或者资源充足)
                        dispatcher.getEventHandler()
                                .handle(
                                        // 创建
ApplicationInitEvent(ApplicationEventType.INIT_APPLICATION)
                                        new ApplicationInitEvent(applicationID,
appAcls, logAggregationContext));
                    }
                } else if (containerTokenIdentifier.getContainerType()
                        == ContainerType.APPLICATION_MASTER) {
                    FlowContext flowContext =
                            getFlowContext(launchContext, applicationID);
```

```
                    if (flowContext != null) {
                        ApplicationImpl application =
                                (ApplicationImpl)
context.getApplications().get(applicationID);

                        // update flowContext reference in ApplicationImpl
                        application.setFlowContext(flowContext);

                        // Required to update state store for recovery.
                        context.getNMStateStore().storeApplication(applicationID,
                                buildAppProto(applicationID, user, credentials,

 container.getLaunchContext().getApplicationACLs(),

 containerTokenIdentifier.getLogAggregationContext(),
                                        flowContext));

                        LOG.info(
                                "Updated application reference with flowContext " +
flowContext
                                        + " for app " + applicationID);
                    } else {
                        LOG.info("TimelineService V2.0 is not enabled. Skipping
updating "
                                + "flowContext for application " + applicationID);
                    }
                }

                this.context.getNMStateStore().storeContainer(containerId,
                        containerTokenIdentifier.getVersion(), containerStartTime,
request);
                // 调用
ApplicationEventDispatcher.handle(ApplicationEventType.INIT_CONTAINER)
                dispatcher.getEventHandler().handle(
                        new ApplicationContainerInitEvent(container));


 this.context.getContainerTokenSecretManager().startContainerSuccessful(
                        containerTokenIdentifier);
                NMAuditLogger.logSuccess(user, AuditConstants.START_CONTAINER,
                        "ContainerManageImpl", applicationID, containerId);
                // TODO launchedContainer misplaced -> doesn't necessarily mean a
container
                // launch. A finished Application will not launch containers.
                metrics.launchedContainer();
                metrics.allocateContainer(containerTokenIdentifier.getResource());
            } else {
                throw new YarnException(
                        "Container start failed as the NodeManager is " +
                        "in the process of shutting down");
            }
        } finally {
```

```
            this.readLock.unlock();
        }
    }
```

4.2.1.2.1 备注说明

```
// 由于 NodeManager 启动 MRAppMaster 容器涉及很多的状态机转换 故文档不暂时 需要的话请阅读源码进行理解
// 所以直接来到 ContainerScheduler.handle(SCHEDULE_CONTAINER) 方法执行调度容器
```

4.2.1.2.2 调度执行 AM 容器(ContainerScheduler.handle(SCHEDULE_CONTAINER))

```
/**
 * The ContainerScheduler manages a collection of runnable containers. It
 * ensures that a container is launched only if all its launch criteria are
 * met. It also ensures that OPPORTUNISTIC containers are killed to make
 * room for GUARANTEED containers.
 */
public class ContainerScheduler extends AbstractService implements
        EventHandler<ContainerSchedulerEvent> {
 /**
     * Handle ContainerSchedulerEvents.
     *
     * @param event ContainerSchedulerEvent.
     */
    @Override
    public void handle(ContainerSchedulerEvent event) {
        switch (event.getType()) {
            // eventType = SCHEDULE_CONTAINER
            case SCHEDULE_CONTAINER:
                // 执行调度容器
                scheduleContainer(event.getContainer());
                break;
            // NOTE: Is sent only after container state has changed to PAUSED...
            case CONTAINER_PAUSED:
                // NOTE: Is sent only after container state has changed to DONE...
            case CONTAINER_COMPLETED:
                onResourcesReclaimed(event.getContainer());
                break;
            case UPDATE_CONTAINER:
                if (event instanceof UpdateContainerSchedulerEvent) {
                    onUpdateContainer((UpdateContainerSchedulerEvent) event);
                } else {
                    LOG.error("Unknown event type on UpdateCOntainer: " +
event.getType());
                }
                break;
            case SHED_QUEUED_CONTAINERS:
                shedQueuedOpportunisticContainers();
                break;
            case RECOVERY_COMPLETED:
```

```
                startPendingContainers(maxOppQueueLength <= 0);
                break;
            default:
                LOG.error("Unknown event arrived at ContainerScheduler: "
                        + event.toString());
        }
    }
}
```

```
@VisibleForTesting
    protected void scheduleContainer(Container container) {
        // true
        boolean isGuaranteedContainer = container.getContainerTokenIdentifier().
                getExecutionType() == ExecutionType.GUARANTEED;

        // Given a guaranteed container, we enqueue it first and then try to start
        // as many queuing guaranteed containers as possible followed by queuing
        // opportunistic containers based on remaining resources available. If the
        // container still stays in the queue afterwards, we need to preempt just
        // enough number of opportunistic containers.
        if (isGuaranteedContainer) {
            // 将调度启动容器放入容器队列等待调度执行
            enqueueContainer(container);

            // When opportunistic container not allowed (which is determined by
            // max-queue length of pending opportunistic containers <= 0), start
            // guaranteed containers without looking at available resources.
            boolean forceStartGuaranteedContainers = (maxOppQueueLength <= 0);
            // 启动容器 (是否强制启动容器 默认 true)
            startPendingContainers(forceStartGuaranteedContainers);

            // if the guaranteed container is queued, we need to preempt
opportunistic
            // containers for make room for it
            if (queuedGuaranteedContainers.containsKey(container.getContainerId()))
{
                reclaimOpportunisticContainerResources(container);
            }
        } else {
            // Given an opportunistic container, we first try to start as many
queuing
            // guaranteed containers as possible followed by queuing opportunistic
            // containers based on remaining resource available, then enqueue the
            // opportunistic container. If the container is enqueued, we do another
            // pass to try to start the newly enqueued opportunistic container.
            startPendingContainers(false);
            boolean containerQueued = enqueueContainer(container);
            // container may not get queued because the max opportunistic container
            // queue length is reached. If so, there is no point doing another pass
            if (containerQueued) {
                startPendingContainers(false);
```

```
            }
        }
    }
```

```
/**
     * Start pending containers in the queue.
     *
     * @param forceStartGuaranteedContaieners When this is true, start guaranteed
     *                                        container without looking at available
resource
     */
    private void startPendingContainers(boolean forceStartGuaranteedContaieners) {
        // Start guaranteed containers that are paused, if resources available.
        // 判断 NodeManager 资源是否够用 (forceStartGuaranteedContaieners = true)
        boolean resourcesAvailable = startContainers(
                queuedGuaranteedContainers.values(),
forceStartGuaranteedContaieners);
        // Start opportunistic containers, if resources available.
        if (resourcesAvailable) {
            startContainers(queuedOpportunisticContainers.values(), false);
        }
    }
```

```
private boolean startContainers(
            Collection<Container> containersToBeStarted, boolean force) {
        // 获取待启动容器
        Iterator<Container> cIter = containersToBeStarted.iterator();
        boolean resourcesAvailable = true;
        while (cIter.hasNext() && resourcesAvailable) {
            // 变量容器
            Container container = cIter.next();
            // 尝试启动容器
            if (tryStartContainer(container, force)) {
                cIter.remove();
            } else {
                resourcesAvailable = false;
            }
        }
        return resourcesAvailable;
    }
```

```
private boolean tryStartContainer(Container container, boolean force) {
        boolean containerStarted = false;
        // call startContainer without checking available resource when force==true
        if (force // 是否强制启动容器
                ||
                resourceAvailableToStartContainer(container) // 判断 NodeManager 资源
是否够用启动容器
        ) {
            // 启动容器
            startContainer(container);
            containerStarted = true;
        }
        return containerStarted;
    }
```

```
    private void startContainer(Container container) {
        // Starting container [container_1684656010852_0005_01_000001]
        LOG.info("Starting container [" + container.getContainerId() + "]");
        // Skip to put into runningContainers and addUtilization when recover
        if (!runningContainers.containsKey(container.getContainerId())) {
            runningContainers.put(container.getContainerId(), container);
            this.utilizationTracker.addContainerResources(container);
        }
        if (container.getContainerTokenIdentifier().getExecutionType() ==
                ExecutionType.OPPORTUNISTIC) {
            this.metrics.startOpportunisticContainer(container.getResource());
        }
        // 发送启动容器事件 调用 ContainerImpl.sendLaunchEvent()
        container.sendLaunchEvent();
    }
```

```
public class ContainerImpl implements Container {
 @SuppressWarnings("unchecked") // dispatcher not typed
    @Override
    public void sendLaunchEvent() {
        if (ContainerState.PAUSED == getContainerState()) {
            dispatcher.getEventHandler().handle(
                    new ContainerResumeEvent(containerId,
                            "Container Resumed as some resources freed up"));
        } else {
            ContainersLauncherEventType launcherEvent =
                    ContainersLauncherEventType.LAUNCH_CONTAINER;
            if (recoveredStatus == RecoveredContainerStatus.LAUNCHED) {
                // try to recover a container that was previously launched
                launcherEvent = ContainersLauncherEventType.RECOVER_CONTAINER;
            } else if (recoveredStatus == RecoveredContainerStatus.PAUSED) {
                launcherEvent =
ContainersLauncherEventType.RECOVER_PAUSED_CONTAINER;
            }
```

```
            containerLaunchStartTime = clock.getTime();
            // 调用
ContainersLauncher.handle(ContainersLauncherEventType.LAUNCH_CONTAINER)
            dispatcher.getEventHandler().handle(
                    new ContainersLauncherEvent(this, launcherEvent));
        }

    }
}
```

4.2.1.2.3 真正启动 AM 容器(调用ContainersLauncher.handle(LAUNCH_CONTAINER))

```
/**
 * The launcher for the containers. This service should be started only after
 * the {@link ResourceLocalizationService} is started as it depends on creation
 * of system directories on the local file-system.
 */
public class ContainersLauncher extends AbstractService
        implements AbstractContainersLauncher {
  @Override
    public void handle(ContainersLauncherEvent event) {
        // TODO: ContainersLauncher launches containers one by one!!
        Container container = event.getContainer();
        ContainerId containerId = container.getContainerId();
        // eventType = LAUNCH_CONTAINER
        switch (event.getType()) {
            case LAUNCH_CONTAINER:
                // 获取启动容器对应的任务 ApplicationId
                Application app =
                        context.getApplications().get(

 containerId.getApplicationAttemptId().getApplicationId());

                // 封装启动容器 ContainerLaunch (是一个 Callable 接口)
                ContainerLaunch launch =
                        new ContainerLaunch(context, getConfig(), dispatcher, exec,
app,
                                event.getContainer(), dirsHandler,
containerManager);

                // 调用 ContainerLaunch.call() 执行启动容器
                containerLauncher.submit(launch);
                running.put(containerId, launch);
                break;
            case RELAUNCH_CONTAINER:
                app = context.getApplications().get(
                        containerId.getApplicationAttemptId().getApplicationId());

                ContainerRelaunch relaunch =
                        new ContainerRelaunch(context, getConfig(), dispatcher,
exec, app,
```

```
                                event.getContainer(), dirsHandler,
containerManager);
                containerLauncher.submit(relaunch);
                running.put(containerId, relaunch);
                break;
            case RECOVER_CONTAINER:
                app = context.getApplications().get(
                        containerId.getApplicationAttemptId().getApplicationId());
                launch = new RecoveredContainerLaunch(context, getConfig(),
dispatcher,
                        exec, app, event.getContainer(), dirsHandler,
containerManager);
                containerLauncher.submit(launch);
                running.put(containerId, launch);
                break;
            case RECOVER_PAUSED_CONTAINER:
                app = context.getApplications().get(
                        containerId.getApplicationAttemptId().getApplicationId());
                launch = new RecoverPausedContainerLaunch(context, getConfig(),
                        dispatcher, exec, app, event.getContainer(), dirsHandler,
                        containerManager);
                containerLauncher.submit(launch);
                break;
            case CLEANUP_CONTAINER:
            case CLEANUP_CONTAINER_FOR_REINIT:
                ContainerLaunch launcher = running.remove(containerId);
                if (launcher == null) {
                    // Container not launched.
                    // triggering KILLING to CONTAINER_CLEANEDUP_AFTER_KILL
transition.
                    dispatcher.getEventHandler().handle(
                            new ContainerExitEvent(containerId,
                                    ContainerEventType.CONTAINER_KILLED_ON_REQUEST,
                                    Shell.WINDOWS ?
ContainerExecutor.ExitCode.FORCE_KILLED.getExitCode() :

 ContainerExecutor.ExitCode.TERMINATED.getExitCode(),
                                    "Container terminated before launch."));
                    return;
                }

                // Cleanup a container whether it is running/killed/completed, so
that
                // no sub-processes are alive.
                try {
                    launcher.cleanupContainer();
                } catch (IOException e) {
                    LOG.warn("Got exception while cleaning container " + containerId
                            + ". Ignoring.");
                }
                break;
            case SIGNAL_CONTAINER:
```

```
                SignalContainersLauncherEvent signalEvent =
                        (SignalContainersLauncherEvent) event;
                ContainerLaunch runningContainer = running.get(containerId);
                if (runningContainer == null) {
                    // Container not launched. So nothing needs to be done.
                    LOG.info("Container " + containerId + " not running, nothing to
signal.");
                    return;
                }

                try {
                    runningContainer.signalContainer(signalEvent.getCommand());
                } catch (IOException e) {
                    LOG.warn("Got exception while signaling container " +
containerId
                            + " with command " + signalEvent.getCommand());
                }
                break;
            case PAUSE_CONTAINER:
                ContainerLaunch launchedContainer = running.get(containerId);
                if (launchedContainer == null) {
                    // Container not launched. So nothing needs to be done.
                    return;
                }

                // Pause the container
                try {
                    launchedContainer.pauseContainer();
                } catch (Exception e) {
                    LOG.info("Got exception while pausing container: " +
                            StringUtils.stringifyException(e));
                }
                break;
            case RESUME_CONTAINER:
                ContainerLaunch launchCont = running.get(containerId);
                if (launchCont == null) {
                    // Container not launched. So nothing needs to be done.
                    return;
                }

                // Resume the container.
                try {
                    launchCont.resumeContainer();
                } catch (Exception e) {
                    LOG.info("Got exception while resuming container: " +
                            StringUtils.stringifyException(e));
                }
                break;
        }
    }
}
```

```java
public class ContainerLaunch implements Callable<Integer> {
    @Override
    @SuppressWarnings("unchecked") // dispatcher not typed
    public Integer call() {
        if (!validateContainerState()) {
            return 0;
        }

        // 获取启动容器的上下文对象
        final ContainerLaunchContext launchContext = container.getLaunchContext();
        // 获取启动容器信息
        ContainerId containerID = container.getContainerId();
        String containerIdStr = containerID.toString();
        // 获取启动容器的命令
        /**
         * 针对 MR 的 AM而言 基本启动 AM 命令模版
         * ${JAVA_HOME}/bin/java
         * -Djava.io.tmpdir=./tmp
         * -Dlog4j.configuration=container-log4j.properties
         * -Dyarn.app.container.log.dir=<LOG_DIR>
         * -Dyarn.app.container.log.filesize=0
         * -Dhadoop.root.logger=INFO,CLA
         * -Dhadoop.root.logfile=syslog
         * -Xmx1024m
         * org.apache.hadoop.mapreduce.v2.app.MRAppMaster
         * 1><LOG_DIR>.stdout
         * 2><LOG_DIR>.stderr
         */
        final List<String> command = launchContext.getCommands();
        int ret = -1;

        Path containerLogDir;
        try {
            Map<Path, List<String>> localResources = getLocalizedResources();

            final String user = container.getUser();
            // /////////////////////////// Variable expansion
            // Before the container script gets written out.
            List<String> newCmds = new ArrayList<String>(command.size());
            String appIdStr = app.getAppId().toString();
            // appIdStr/containerIdStr (相对路径)
            String relativeContainerLogDir = ContainerLaunch
                    .getRelativeContainerLogDir(appIdStr, containerIdStr);
            // 容器日志目录 (也即在本地创建日志目录)
            containerLogDir =
                    dirsHandler.getLogPathForWrite(relativeContainerLogDir, false);
            recordContainerLogDir(containerID, containerLogDir.toString());

            for (String str : command) {
                // TODO: Should we instead work via symlinks without this grammar?
                // 扩展环境命令 (也即把 <LOG_DIR> 替换成容器日志目录 containerLogDir)
                newCmds.add(expandEnvironment(str, containerLogDir));
```

```
            }
            // 重新更为启动容器命令
            launchContext.setCommands(newCmds);

            // 环境所有的环境变量
            Map<String, String> environment = expandAllEnvironmentVars(
                    launchContext, containerLogDir);
            // /////////////////////////// End of variable expansion

            // Use this to track variables that are added to the environment by nm.
            LinkedHashSet<String> nmEnvVars = new LinkedHashSet<String>();

            // 获取本地文件系统
            FileContext lfs = FileContext.getLocalFSFileContext();

            // 获取容器脚本路径 (xxx/launch_container)
            Path nmPrivateContainerScriptPath = dirsHandler.getLocalPathForWrite(
                    getContainerPrivateDir(appIdStr, containerIdStr) +
Path.SEPARATOR
                            + CONTAINER_SCRIPT);
            // 获取私有 Token 路径
            Path nmPrivateTokensPath = dirsHandler.getLocalPathForWrite(
                    getContainerPrivateDir(appIdStr, containerIdStr) +
Path.SEPARATOR
                            + String.format(ContainerLocalizer.TOKEN_FILE_NAME_FMT,
                            containerIdStr));
            // 获取类JAR路径
            Path nmPrivateClasspathJarDir = dirsHandler.getLocalPathForWrite(
                    getContainerPrivateDir(appIdStr, containerIdStr));

            // Select the working directory for the container
            // 获取容器工作目录
            Path containerWorkDir = deriveContainerWorkDir();
            recordContainerWorkDir(containerID, containerWorkDir.toString());

            // 获取启动容器进程 PID 文件路径
            String pidFileSubpath = getPidFileSubpath(appIdStr, containerIdStr);
            // pid file should be in nm private dir so that it is not
            // accessible by users
            pidFilePath = dirsHandler.getLocalPathForWrite(pidFileSubpath);

            // 获取相关文件路径目录
            List<String> localDirs = dirsHandler.getLocalDirs();
            List<String> localDirsForRead = dirsHandler.getLocalDirsForRead();
            List<String> logDirs = dirsHandler.getLogDirs();
            List<String> filecacheDirs = getNMFilecacheDirs(localDirsForRead);
            List<String> userLocalDirs = getUserLocalDirs(localDirs);
            List<String> containerLocalDirs = getContainerLocalDirs(localDirs);
            List<String> containerLogDirs = getContainerLogDirs(logDirs);
            List<String> userFilecacheDirs = getUserFilecacheDirs(localDirsForRead);
            List<String> applicationLocalDirs = getApplicationLocalDirs(localDirs,
appIdStr);
```

```
            if (!dirsHandler.areDisksHealthy()) {
                ret = ContainerExitStatus.DISKS_FAILED;
                throw new IOException("Most of the disks failed. "
                        + dirsHandler.getDisksHealthReport(false));
            }
            List<Path> appDirs = new ArrayList<Path>(localDirs.size());
            for (String localDir : localDirs) {
                Path usersdir = new Path(localDir, ContainerLocalizer.USERCACHE);
                Path userdir = new Path(usersdir, user);
                Path appsdir = new Path(userdir, ContainerLocalizer.APPCACHE);
                appDirs.add(new Path(appsdir, appIdStr));
            }

            // Set the token location too.
            // 设置本地 Tokent (xxx/container_tokens)
            addToEnvMap(environment, nmEnvVars,
                    ApplicationConstants.CONTAINER_TOKEN_FILE_ENV_NAME,
                    new Path(containerWorkDir,
                            FINAL_CONTAINER_TOKENS_FILE).toUri().getPath());

            // /////////// Write out the container-script in the nmPrivate space.
            // 写启动容器脚本到文件 (launch_container)
            try (DataOutputStream containerScriptOutStream =
                         lfs.create(nmPrivateContainerScriptPath,
                                 EnumSet.of(CREATE, OVERWRITE))) {
                // Sanitize the container's environment
                sanitizeEnv(environment, containerWorkDir, appDirs, userLocalDirs,
                        containerLogDirs, localResources, nmPrivateClasspathJarDir,
                        nmEnvVars);

                // 准备容器
                prepareContainer(localResources, containerLocalDirs);

                // Write out the environment
                // 写相关环境信息到 launch_container 脚本
                exec.writeLaunchEnv(containerScriptOutStream, environment,
                        localResources, launchContext.getCommands(),
                        containerLogDir, user, nmEnvVars);
            }
            // /////////// End of writing out container-script

            // /////////// Write out the container-tokens in the nmPrivate space.
            // 写容器 Token 消息到文件
            try (DataOutputStream tokensOutStream =
                         lfs.create(nmPrivateTokensPath, EnumSet.of(CREATE,
OVERWRITE))) {
                Credentials creds = container.getCredentials();
                creds.writeTokenStorageToStream(tokensOutStream);
            }
            // /////////// End of writing out container-tokens
```

```
            // 执行脚本启动容器
            ret = launchContainer(new ContainerStartContext.Builder()
                    .setContainer(container)
                    .setLocalizedResources(localResources)
                    .setNmPrivateContainerScriptPath(nmPrivateContainerScriptPath)
                    .setNmPrivateTokensPath(nmPrivateTokensPath)
                    .setUser(user)
                    .setAppId(appIdStr)
                    .setContainerWorkDir(containerWorkDir)
                    .setLocalDirs(localDirs)
                    .setLogDirs(logDirs)
                    .setFilecacheDirs(filecacheDirs)
                    .setUserLocalDirs(userLocalDirs)
                    .setContainerLocalDirs(containerLocalDirs)
                    .setContainerLogDirs(containerLogDirs)
                    .setUserFilecacheDirs(userFilecacheDirs)
                    .setApplicationLocalDirs(applicationLocalDirs).build());
        } catch (ConfigurationException e) {
            LOG.error("Failed to launch container due to configuration error.", e);
            dispatcher.getEventHandler().handle(new ContainerExitEvent(
                    containerID, ContainerEventType.CONTAINER_EXITED_WITH_FAILURE,
ret,
                    e.getMessage()));
            // Mark the node as unhealthy
            context.getNodeStatusUpdater().reportException(e);
            return ret;
        } catch (Throwable e) {
            LOG.warn("Failed to launch container.", e);
            dispatcher.getEventHandler().handle(new ContainerExitEvent(
                    containerID, ContainerEventType.CONTAINER_EXITED_WITH_FAILURE,
ret,
                    e.getMessage()));
            return ret;
        } finally {
            setContainerCompletedStatus(ret);
        }

        handleContainerExitCode(ret, containerLogDir);
        return ret;
    }
}
```

五 启动 MRAppMaster 进程 (MR 任务的 ApplicationMaster)

```
/**
 * The Map-Reduce Application Master.
 * The state machine is encapsulated in the implementation of Job interface.
 * All state changes happens via Job interface. Each event
 * results in a Finite State Transition in Job.
```

```
 *
 * MR AppMaster is the composition of loosely coupled services. The services
 * interact with each other via events. The components resembles the
 * Actors model. The component acts on received event and send out the
 * events to other components.
 * This keeps it highly concurrent with no or minimal synchronization needs.
 *
 * The events are dispatched by a central Dispatch mechanism. All components
 * register to the Dispatcher.
 *
 * The information is shared across different components using AppContext.
 */

@SuppressWarnings("rawtypes")
public class MRAppMaster extends CompositeService {
 public static void main(String[] args) {
        try {
            mainStarted = true;
            Thread.setDefaultUncaughtExceptionHandler(new
YarnUncaughtExceptionHandler());
            // 从当前 NodeManager 的节点获取当前启动容器的环境变量 key = CONTAINER_ID
            /**
             * 这些环境变量存储在 launch_container 脚本里
             * 以 export key = value 形式暴露
             */
            String containerIdStr =
                    System.getenv(Environment.CONTAINER_ID.name());
            String nodeHostString = System.getenv(Environment.NM_HOST.name());
            String nodePortString = System.getenv(Environment.NM_PORT.name());
            String nodeHttpPortString =
                    System.getenv(Environment.NM_HTTP_PORT.name());
            String appSubmitTimeStr =
                    System.getenv(ApplicationConstants.APP_SUBMIT_TIME_ENV);
            validateInputParam(containerIdStr,
                    Environment.CONTAINER_ID.name());
            validateInputParam(nodeHostString, Environment.NM_HOST.name());
            validateInputParam(nodePortString, Environment.NM_PORT.name());
            validateInputParam(nodeHttpPortString,
                    Environment.NM_HTTP_PORT.name());
            validateInputParam(appSubmitTimeStr,
                    ApplicationConstants.APP_SUBMIT_TIME_ENV);

            ContainerId containerId = ContainerId.fromString(containerIdStr);
            ApplicationAttemptId applicationAttemptId =
                    containerId.getApplicationAttemptId();
            if (applicationAttemptId != null) {
                CallerContext.setCurrent(new CallerContext.Builder(
                        "mr_appmaster_" + applicationAttemptId.toString()).build());
            }
            long appSubmitTime = Long.parseLong(appSubmitTimeStr);

            // 创建 MRAppMaster
```

```
            MRAppMaster appMaster =
                    new MRAppMaster(
                            applicationAttemptId,
                            containerId,
                            nodeHostString,
                            Integer.parseInt(nodePortString),
                            Integer.parseInt(nodeHttpPortString),
                            appSubmitTime);

            ShutdownHookManager.get().addShutdownHook(
                    new MRAppMasterShutdownHook(appMaster), SHUTDOWN_HOOK_PRIORITY);
            JobConf conf = new JobConf(new YarnConfiguration());
            conf.addResource(new Path(MRJobConfig.JOB_CONF_FILE));

            MRWebAppUtil.initialize(conf);
            // log the system properties
            String systemPropsToLog = MRApps.getSystemPropertiesToLog(conf);
            if (systemPropsToLog != null) {
                /**
                 * /************************************************************
                 * [system properties]
                 * os.name: Linux
                 * os.version: 3.10.0-1062.el7.x86_64
                 * java.home: /opt/app/jdk1.8.0_212/jre
                 * java.runtime.version: 1.8.0_212-b10
                 * java.vendor: Oracle Corporation
                 * java.version: 1.8.0_212
                 * java.vm.name: Java HotSpot(TM) 64-Bit Server VM
```

```
                * java.class.path: /opt/app/hadoop-3.1.3/data/nm-local-
dir/usercache/tanbs/appcache/application_1684656010852_0006/container_1684656010852_
0006_01_000001:/opt/app/hadoop-3.1.3/etc/hadoop:/opt/app/hadoop-
3.1.3/share/hadoop/common/hadoop-common-3.1.3-tests.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/hadoop-common-3.1.3.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/hadoop-kms-3.1.3.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/hadoop-nfs-3.1.3.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/jetty-io-9.3.24.v20180605.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/accessors-smart-1.2.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/jcip-annotations-1.0-1.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/animal-sniffer-annotations-1.17.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/netty-3.10.5.Final.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/asm-5.0.4.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/jersey-core-1.19.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/audience-annotations-0.5.0.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/nimbus-jose-jwt-4.41.1.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/avro-1.7.7.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/jetty-security-9.3.24.v20180605.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/checker-qual-2.5.2.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/jetty-server-9.3.24.v20180605.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/commons-beanutils-1.9.3.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/paranamer-2.3.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/commons-cli-1.2.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/jsr311-api-1.1.1.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/commons-codec-1.11.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/jersey-json-1.19.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/commons-collections-3.2.2.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/jul-to-slf4j-1.7.25.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/commons-compress-1.18.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/jersey-servlet-1.19.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/commons-configuration2-2.1.1.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/protobuf-java-2.5.0.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/commons-io-2.5.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/re2j-1.1.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/commons-lang-2.6.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/kerb-admin-1.0.1.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/commons-lang3-3.4.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/kerb-client-1.0.1.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/commons-logging-1.1.3.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/kerb-common-1.0.1.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/commons-math3-3.1.1.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/slf4j-api-1.7.25.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/commons-net-3.6.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/kerb-core-1.0.1.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/curator-client-2.13.0.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/kerb-crypto-1.0.1.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/curator-framework-2.13.0.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/kerb-identity-1.0.1.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/curator-recipes-2.13.0.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/jersey-server-1.19.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/error_prone_annotations-2.2.0.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/kerb-server-1.0.1.jar:/opt/app/hadoop-
```

```
3.1.3/share/hadoop/common/lib/failureaccess-1.0.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/slf4j-log4j12-1.7.25.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/gson-2.2.4.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/snappy-java-1.0.5.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/guava-27.0-jre.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/kerb-simplekdc-1.0.1.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/hadoop-annotations-3.1.3.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/kerb-util-1.0.1.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/hadoop-auth-3.1.3.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/jsch-0.1.54.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/htrace-core4-4.1.0-incubating.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/stax2-api-3.1.4.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/httpclient-4.5.2.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/token-provider-1.0.1.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/httpcore-4.4.4.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/kerby-asn1-1.0.1.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/j2objc-annotations-1.1.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/jettison-1.1.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/jackson-annotations-2.7.8.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/kerby-config-1.0.1.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/jackson-core-2.7.8.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/kerby-pkix-1.0.1.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/jackson-core-asl-1.9.13.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/kerby-util-1.0.1.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/jackson-databind-2.7.8.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/kerby-xdr-1.0.1.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/jackson-jaxrs-1.9.13.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/jetty-http-9.3.24.v20180605.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/jackson-mapper-asl-1.9.13.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/jackson-xc-1.9.13.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/log4j-1.2.17.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/javax.servlet-api-3.1.0.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/woodstox-core-5.0.3.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/jaxb-api-2.2.11.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/metrics-core-3.2.4.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/jaxb-impl-2.2.3-1.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/jetty-servlet-9.3.24.v20180605.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/json-smart-2.3.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/jetty-util-9.3.24.v20180605.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/jsp-api-2.1.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/jetty-webapp-9.3.24.v20180605.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/jsr305-3.0.0.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/jetty-xml-9.3.24.v20180605.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/listenablefuture-9999.0-empty-to-avoid-conflict-with-
guava.jar:/opt/app/hadoop-3.1.3/share/hadoop/common/lib/zookeeper-
3.4.13.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/hadoop-hdfs-3.1.3-
tests.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/hadoop-hdfs-
3.1.3.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/hadoop-hdfs-client-3.1.3-
tests.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/hadoop-hdfs-client-
3.1.3.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/hadoop-hdfs-httpfs-
3.1.3.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/hadoop-hdfs-native-client-3.1.3-
tests.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/hadoop-hdfs-native-client-
```

```
3.1.3.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/hadoop-hdfs-nfs-
3.1.3.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/hadoop-hdfs-rbf-3.1.3-
tests.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/hadoop-hdfs-rbf-
3.1.3.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/jetty-http-
9.3.24.v20180605.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/accessors-smart-
1.2.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/jaxb-impl-2.2.3-
1.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/animal-sniffer-annotations-
1.17.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/log4j-
1.2.17.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/asm-
5.0.4.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/jcip-annotations-1.0-
1.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/audience-annotations-
0.5.0.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/netty-
3.10.5.Final.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/avro-
1.7.7.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/jetty-io-
9.3.24.v20180605.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/checker-qual-
2.5.2.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/jetty-security-
9.3.24.v20180605.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/commons-beanutils-
1.9.3.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/netty-all-
4.0.52.Final.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/commons-cli-
1.2.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/json-smart-
2.3.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/commons-codec-
1.11.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/jersey-core-
1.19.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/commons-collections-
3.2.2.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/jsr305-
3.0.0.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/commons-compress-
1.18.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/jersey-server-
1.19.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/commons-configuration2-
2.1.1.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/jsr311-api-
1.1.1.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/commons-daemon-
1.0.13.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/nimbus-jose-jwt-
4.41.1.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/commons-io-
2.5.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/okhttp-
2.7.5.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/commons-lang-
2.6.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/kerb-admin-
1.0.1.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/commons-lang3-
3.4.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/kerb-client-
1.0.1.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/commons-logging-
1.1.3.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/kerb-common-
1.0.1.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/commons-math3-
3.1.1.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/okio-
1.6.0.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/commons-net-
3.6.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/kerb-core-
1.0.1.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/curator-client-
2.13.0.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/kerb-crypto-
1.0.1.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/curator-framework-
2.13.0.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/kerb-identity-
1.0.1.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/curator-recipes-
2.13.0.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/jersey-json-
1.19.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/error_prone_annotations-
2.2.0.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/kerb-server-
1.0.1.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/failureaccess-
1.0.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/paranamer-
```

```
2.3.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/gson-2.2.4.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/protobuf-java-2.5.0.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/guava-27.0-jre.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/kerb-simplekdc-1.0.1.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/hadoop-annotations-3.1.3.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/kerb-util-1.0.1.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/hadoop-auth-3.1.3.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/jetty-webapp-9.3.24.v20180605.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/htrace-core4-4.1.0-incubating.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/re2j-1.1.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/httpclient-4.5.2.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/snappy-java-1.0.5.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/httpcore-4.4.4.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/kerby-asn1-1.0.1.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/j2objc-annotations-1.1.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/jersey-servlet-1.19.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/jackson-annotations-2.7.8.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/kerby-config-1.0.1.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/jackson-core-2.7.8.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/kerby-pkix-1.0.1.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/jackson-core-asl-1.9.13.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/kerby-util-1.0.1.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/jackson-databind-2.7.8.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/kerby-xdr-1.0.1.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/jackson-jaxrs-1.9.13.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/jettison-1.1.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/jackson-mapper-asl-1.9.13.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/leveldbjni-all-1.8.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/jackson-xc-1.9.13.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/javax.servlet-api-3.1.0.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/stax2-api-3.1.4.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/jaxb-api-2.2.11.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/jetty-server-9.3.24.v20180605.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/jetty-xml-9.3.24.v20180605.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/jetty-servlet-9.3.24.v20180605.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/jsch-0.1.54.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/jetty-util-9.3.24.v20180605.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/json-simple-1.1.1.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/jetty-util-ajax-9.3.24.v20180605.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/listenablefuture-9999.0-empty-to-avoid-conflict-with-
guava.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/token-provider-
1.0.1.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/woodstox-core-
5.0.3.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/zookeeper-
3.4.13.jar:/opt/app/hadoop-3.1.3/share/hadoop/yarn/hadoop-yarn-api-
3.1.3.jar:/opt/app/hadoop-3.1.3/share/hadoop/yarn/hadoop-yarn-applications-
distributedshell-3.1.3.jar:/opt/app/hadoop-3.1.3/share/hadoop/yarn/hadoop-yarn-
applications-unmanaged-am-launcher-3.1.3.jar:/opt/app/hadoop-
3.1.3/share/hadoop/yarn/hadoop-yarn-client-3.1.3.jar:/opt/app/hadoop-
3.1.3/share/hadoop/yarn/hadoop-yarn-common-3.1.3.jar:/opt/app/hadoop-
3.1.3/share/hadoop/yarn/hadoop-yarn-registry-3.1.3.jar:/opt/app/hadoop-
3.1.3/share/hadoop/yarn/hadoop-yarn-server-applicationhistoryservice-
3.1.3.jar:/opt/app/hadoop-3.1.3/share/hadoop/yarn/hadoop-yarn-server-common-
```

```
3.1.3.jar:/opt/app/hadoop-3.1.3/share/hadoop/yarn/hadoop-yarn-server-nodemanager-
3.1.3.jar:/opt/app/hadoop-3.1.3/share/hadoop/yarn/hadoop-yarn-server-
resourcemanager-3.1.3.jar:/opt/app/hadoop-3.1.3/share/hadoop/yarn/hadoop-yarn-
server-router-3.1.3.jar:/opt/app/hadoop-3.1.3/share/hadoop/yarn/hadoop-yarn-server-
sharedcachemanager-3.1.3.jar:/opt/app/hadoop-3.1.3/share/hadoop/yarn/hadoop-yarn-
server-tests-3.1.3.jar:/opt/app/hadoop-3.1.3/share/hadoop/yarn/hadoop-yarn-server-
timeline-pluginstorage-3.1.3.jar:/opt/app/hadoop-3.1.3/share/hadoop/yarn/hadoop-
yarn-server-web-proxy-3.1.3.jar:/opt/app/hadoop-3.1.3/share/hadoop/yarn/hadoop-yarn-
services-api-3.1.3.jar:/opt/app/hadoop-3.1.3/share/hadoop/yarn/hadoop-yarn-services-
core-3.1.3.jar:/opt/app/hadoop-3.1.3/share/hadoop/yarn/lib/HikariCP-java7-
2.4.12.jar:/opt/app/hadoop-3.1.3/share/hadoop/yarn/lib/aopalliance-
1.0.jar:/opt/app/hadoop-3.1.3/share/hadoop/yarn/lib/dnsjava-
2.1.7.jar:/opt/app/hadoop-3.1.3/share/hadoop/yarn/lib/ehcache-
3.3.1.jar:/opt/app/hadoop-3.1.3/share/hadoop/yarn/lib/fst-2.50.jar:/opt/app/hadoop-
3.1.3/share/hadoop/yarn/lib/geronimo-jcache_1.0_spec-1.0-alpha-
1.jar:/opt/app/hadoop-3.1.3/share/hadoop/yarn/lib/guice-4.0.jar:/opt/app/hadoop-
3.1.3/share/hadoop/yarn/lib/guice-servlet-4.0.jar:/opt/app/hadoop-
3.1.3/share/hadoop/yarn/lib/jackson-jaxrs-base-2.7.8.jar:/opt/app/hadoop-
3.1.3/share/hadoop/yarn/lib/jackson-jaxrs-json-provider-2.7.8.jar:/opt/app/hadoop-
3.1.3/share/hadoop/yarn/lib/jackson-module-jaxb-annotations-
2.7.8.jar:/opt/app/hadoop-3.1.3/share/hadoop/yarn/lib/java-util-
1.9.0.jar:/opt/app/hadoop-3.1.3/share/hadoop/yarn/lib/javax.inject-
1.jar:/opt/app/hadoop-3.1.3/share/hadoop/yarn/lib/jersey-client-
1.19.jar:/opt/app/hadoop-3.1.3/share/hadoop/yarn/lib/jersey-guice-
1.19.jar:/opt/app/hadoop-3.1.3/share/hadoop/yarn/lib/json-io-
2.5.1.jar:/opt/app/hadoop-3.1.3/share/hadoop/yarn/lib/metrics-core-
3.2.4.jar:/opt/app/hadoop-3.1.3/share/hadoop/yarn/lib/mssql-jdbc-
6.2.1.jre7.jar:/opt/app/hadoop-3.1.3/share/hadoop/yarn/lib/objenesis-
1.0.jar:/opt/app/hadoop-3.1.3/share/hadoop/yarn/lib/snakeyaml-
1.16.jar:/opt/app/hadoop-3.1.3/share/hadoop/yarn/lib/swagger-annotations-
1.5.4.jar:/opt/app/hadoop-3.1.3/share/hadoop/mapreduce/hadoop-mapreduce-client-app-
3.1.3.jar:/opt/app/hadoop-3.1.3/share/hadoop/mapreduce/hadoop-mapreduce-client-
common-3.1.3.jar:/opt/app/hadoop-3.1.3/share/hadoop/mapreduce/hadoop-mapreduce-
client-core-3.1.3.jar:/opt/app/hadoop-3.1.3/share/hadoop/mapreduce/hadoop-mapreduce-
client-hs-3.1.3.jar:/opt/app/hadoop-3.1.3/share/hadoop/mapreduce/hadoop-mapreduce-
client-hs-plugins-3.1.3.jar:/opt/app/hadoop-3.1.3/share/hadoop/mapreduce/hadoop-
mapreduce-client-jobclient-3.1.3-tests.jar:/opt/app/hadoop-
3.1.3/share/hadoop/mapreduce/hadoop-mapreduce-client-jobclient-
3.1.3.jar:/opt/app/hadoop-3.1.3/share/hadoop/mapreduce/hadoop-mapreduce-client-
nativetask-3.1.3.jar:/opt/app/hadoop-3.1.3/share/hadoop/mapreduce/hadoop-mapreduce-
client-shuffle-3.1.3.jar:/opt/app/hadoop-3.1.3/share/hadoop/mapreduce/hadoop-
mapreduce-client-uploader-3.1.3.jar:/opt/app/hadoop-
3.1.3/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.1.3.jar:/opt/app/hadoop-
3.1.3/share/hadoop/mapreduce/lib/hamcrest-core-1.3.jar:/opt/app/hadoop-
3.1.3/share/hadoop/mapreduce/lib/junit-
4.11.jar:job.jar/job.jar:job.jar/classes/:job.jar/lib/*:/opt/app/hadoop-
3.1.3/data/nm-local-
dir/usercache/tanbs/appcache/application_1684656010852_0006/container_1684656010852_
0006_01_000001/job.jar
                * java.io.tmpdir: /opt/app/hadoop-3.1.3/data/nm-local-
dir/usercache/tanbs/appcache/application_1684656010852_0006/container_1684656010852_
0006_01_000001/tmp
```

```
                 * user.dir: /opt/app/hadoop-3.1.3/data/nm-local-
dir/usercache/tanbs/appcache/application_1684656010852_0006/container_1684656010852_
0006_01_000001
                 * user.name: tanbs
                 * ************************************************************
                 */
                LOG.info(systemPropsToLog);
            }

            String jobUserName = System
                    .getenv(ApplicationConstants.Environment.USER.name());
            conf.set(MRJobConfig.USER_NAME, jobUserName);

            // 初始化并启动 AM
            initAndStartAppMaster(appMaster, conf, jobUserName);
        } catch (Throwable t) {
            LOG.error("Error starting MRAppMaster", t);
            ExitUtil.terminate(1, t);
        }
    }
}
```

5.1 创建 MRAppMaster 组合服务

```
/**
 * The Map-Reduce Application Master.
 * The state machine is encapsulated in the implementation of Job interface.
 * All state changes happens via Job interface. Each event
 * results in a Finite State Transition in Job.
 * <p>
 * MR AppMaster is the composition of loosely coupled services. The services
 * interact with each other via events. The components resembles the
 * Actors model. The component acts on received event and send out the
 * events to other components.
 * This keeps it highly concurrent with no or minimal synchronization needs.
 * <p>
 * The events are dispatched by a central Dispatch mechanism. All components
 * register to the Dispatcher.
 * <p>
 * The information is shared across different components using AppContext.
 */

@SuppressWarnings("rawtypes")
public class MRAppMaster extends CompositeService {
    public MRAppMaster(ApplicationAttemptId applicationAttemptId,
                       ContainerId containerId, String nmHost, int nmPort, int
nmHttpPort,
                       long appSubmitTime) {
        // 往下追
        this(applicationAttemptId, containerId, nmHost, nmPort, nmHttpPort,
                SystemClock.getInstance(), appSubmitTime);
```

```
    }

    public MRAppMaster(ApplicationAttemptId applicationAttemptId,
                       ContainerId containerId, String nmHost, int nmPort, int
nmHttpPort,
                       Clock clock, long appSubmitTime) {
        super(MRAppMaster.class.getName());
        this.clock = clock;
        this.startTime = clock.getTime();
        this.appSubmitTime = appSubmitTime;
        this.appAttemptID = applicationAttemptId;
        this.containerID = containerId;
        this.nmHost = nmHost;
        this.nmPort = nmPort;
        this.nmHttpPort = nmHttpPort;
        this.metrics = MRAppMetrics.create();
        logSyncer = TaskLog.createLogSyncer();
        // Created MRAppMaster for application appattempt_1684656010852_0006_000001
        LOG.info("Created MRAppMaster for application " + applicationAttemptId);
    }
}
```

5.2 初始化并启动 AM

```
protected static void initAndStartAppMaster(final MRAppMaster appMaster,
                                            final JobConf conf, String
jobUserName) throws IOException,
            InterruptedException {
        UserGroupInformation.setConfiguration(conf);
        // MAPREDUCE-6565: need to set configuration for SecurityUtil.
        SecurityUtil.setConfiguration(conf);
        // Security framework already loaded the tokens into current UGI, just use
        // them
        Credentials credentials =
                UserGroupInformation.getCurrentUser().getCredentials();

        // Executing with tokens:
        // [Kind: YARN_AM_RM_TOKEN, Service: ,
        // Ident: (appAttemptId { application_id
        // { id: 6 cluster_timestamp: 1684656010852 } attemptId: 1 } keyId:
-1993740727)]
        LOG.info("Executing with tokens: {}", credentials.getAllTokens());

        UserGroupInformation appMasterUgi = UserGroupInformation
                .createRemoteUser(jobUserName);
        appMasterUgi.addCredentials(credentials);

        // Now remove the AM->RM token so tasks don't have it
        Iterator<Token<?>> iter = credentials.getAllTokens().iterator();
        while (iter.hasNext()) {
            Token<?> token = iter.next();
```

```
            if (token.getKind().equals(AMRMTokenIdentifier.KIND_NAME)) {
                iter.remove();
            }
        }
        conf.getCredentials().addAll(credentials);
        appMasterUgi.doAs(new PrivilegedExceptionAction<Object>() {
            @Override
            public Object run() throws Exception {
                // 初始化 MRAppMaster (调用 MRAppMaster.serviceInit())
                appMaster.init(conf);
                // 启动 MRAppMaster (调用 MRAppMaster.serviceStart())
                appMaster.start();
                if (appMaster.errorHappenedShutDown) {
                    throw new IOException("Was asked to shut down.");
                }
                return null;
            }
        });
    }
```

5.2.1 初始化 MRAppMaster (调用 MRAppMaster.serviceInit())

```
@Override
    protected void serviceInit(final Configuration conf) throws Exception {
        // create the job classloader if enabled
        // 如果开启了任务类加载器 默认不开启
        createJobClassLoader(conf);

        initJobCredentialsAndUGI(conf);

        // 创建并添加 AsyncDispatcher 服务
        dispatcher = createDispatcher();
        addIfService(dispatcher);

        // 创建并添加任务完成监控服务 TaskAttemptFinishingMonitor
        taskAttemptFinishingMonitor =
createTaskAttemptFinishingMonitor(dispatcher.getEventHandler());
        addIfService(taskAttemptFinishingMonitor);

        // 创建 AM 运行上下文对象 RunningAppContext
        context = new RunningAppContext(conf, taskAttemptFinishingMonitor);

        // Job name is the same as the app name util we support DAG of jobs
        // for an app later
        // 获取任务名称
        appName = conf.get(MRJobConfig.JOB_NAME, "<missing app name>");

        // 设置 key = mapreduce.job.application.attempt.id value =
ApplicationAttemptId
        conf.setInt(MRJobConfig.APPLICATION_ATTEMPT_ID,
appAttemptID.getAttemptId());
```

```
        newApiCommitter = false;
        jobId = MRBuilderUtils.newJobId(appAttemptID.getApplicationId(),
                appAttemptID.getApplicationId().getId());
        int numReduceTasks = conf.getInt(MRJobConfig.NUM_REDUCES, 0);
        if ((numReduceTasks > 0 &&
                conf.getBoolean("mapred.reducer.new-api", false)) ||
                (numReduceTasks == 0 &&
                        conf.getBoolean("mapred.mapper.new-api", false))) {
            newApiCommitter = true;
            // Using mapred newApiCommitter.
            LOG.info("Using mapred newApiCommitter.");
        }

        boolean copyHistory = false;
        // 创建 FileOutputCommitter
        committer = createOutputCommitter(conf);
        try {
            String user = UserGroupInformation.getCurrentUser().getShortUserName();
            Path stagingDir = MRApps.getStagingAreaDir(conf, user);
            FileSystem fs = getFileSystem(conf);

            boolean stagingExists = fs.exists(stagingDir);
            Path startCommitFile = MRApps.getStartJobCommitFile(conf, user, jobId);
            boolean commitStarted = fs.exists(startCommitFile);
            Path endCommitSuccessFile = MRApps.getEndJobCommitSuccessFile(conf,
user, jobId);
            boolean commitSuccess = fs.exists(endCommitSuccessFile);
            Path endCommitFailureFile = MRApps.getEndJobCommitFailureFile(conf,
user, jobId);
            boolean commitFailure = fs.exists(endCommitFailureFile);

            if (!stagingExists) {
                isLastAMRetry = true;
                LOG.info("Attempt num: " + appAttemptID.getAttemptId() +
                        " is last retry: " + isLastAMRetry +
                        " because the staging dir doesn't exist.");
                errorHappenedShutDown = true;
                forcedState = JobStateInternal.ERROR;
                shutDownMessage = "Staging dir does not exist " + stagingDir;
                LOG.error(shutDownMessage);
            } else if (commitStarted) {
                //A commit was started so this is the last time, we just need to
know
                // what result we will use to notify, and how we will unregister
                errorHappenedShutDown = true;
                isLastAMRetry = true;
                LOG.info("Attempt num: " + appAttemptID.getAttemptId() +
                        " is last retry: " + isLastAMRetry +
                        " because a commit was started.");
                copyHistory = true;
                if (commitSuccess) {
```

```
                    shutDownMessage =
                            "Job commit succeeded in a prior MRAppMaster attempt " +
                                    "before it crashed. Recovering.";
                    forcedState = JobStateInternal.SUCCEEDED;
                } else if (commitFailure) {
                    shutDownMessage =
                            "Job commit failed in a prior MRAppMaster attempt " +
                                    "before it crashed. Not retrying.";
                    forcedState = JobStateInternal.FAILED;
                } else {
                    if (isCommitJobRepeatable()) {
                        // cleanup previous half done commits if committer supports
                        // repeatable job commit.
                        errorHappenedShutDown = false;
                        cleanupInterruptedCommit(conf, fs, startCommitFile);
                    } else {
                        //The commit is still pending, commit error
                        shutDownMessage =
                                "Job commit from a prior MRAppMaster attempt is " +
                                        "potentially in progress. Preventing
multiple commit executions";
                        forcedState = JobStateInternal.ERROR;
                    }
                }
            }
        } catch (IOException e) {
            throw new YarnRuntimeException("Error while initializing", e);
        }

        // 默认 false
        if (errorHappenedShutDown) {
            NoopEventHandler eater = new NoopEventHandler();
            //We do not have a JobEventDispatcher in this path
            dispatcher.register(JobEventType.class, eater);

            EventHandler<JobHistoryEvent> historyService = null;
            if (copyHistory) {
                historyService =
                        createJobHistoryHandler(context);

 dispatcher.register(org.apache.hadoop.mapreduce.jobhistory.EventType.class,
                        historyService);
            } else {

 dispatcher.register(org.apache.hadoop.mapreduce.jobhistory.EventType.class,
                        eater);
            }

            if (copyHistory) {
                // Now that there's a FINISHING state for application on RM to give
AMs
```

```
                // plenty of time to clean up after unregister it's safe to clean
staging
                // directory after unregistering with RM. So, we start the staging-
dir
                // cleaner BEFORE the ContainerAllocator so that on shut-down,
                // ContainerAllocator unregisters first and then the staging-dir
cleaner
                // deletes staging directory.
                addService(createStagingDirCleaningService());
            }

            // service to allocate containers from RM (if non-uber) or to fake it
(uber)
            containerAllocator = createContainerAllocator(null, context);
            addIfService(containerAllocator);
            dispatcher.register(ContainerAllocator.EventType.class,
containerAllocator);

            if (copyHistory) {
                // Add the JobHistoryEventHandler last so that it is properly
stopped first.
                // This will guarantee that all history-events are flushed before AM
goes
                // ahead with shutdown.
                // Note: Even though JobHistoryEventHandler is started last, if any
                // component creates a JobHistoryEvent in the meanwhile, it will be
just be
                // queued inside the JobHistoryEventHandler
                addIfService(historyService);

                JobHistoryCopyService cpHist = new
JobHistoryCopyService(appAttemptID,
                        dispatcher.getEventHandler());
                addIfService(cpHist);
            }
        } else {

            //service to handle requests from JobClient
            // 创建 MRClientService 服务 该服务是与 JobClient 客户端交互(也即提交任务的客户
端)
            // JobClient 持续获取提交任务的状态信息
            // 通讯协议接口为 MRClientProtocol
            clientService = createClientService(context);
            // Init ClientService separately so that we stop it separately, since
this
            // service needs to wait some time before it stops so clients can know
the
            // final states
            // 初始化 MRClientService
            // (调用 MRClientService的 父类 AbstractService.serviceInit())
            clientService.init(conf);
```

```
            // 创建容器申请器 ContainerAllocatorRouter
            containerAllocator = createContainerAllocator(clientService, context);

            //service to handle the output committer
            // 创建并添加 CommitterEventHandler 服务
            committerEventHandler = createCommitterEventHandler(context, committer);
            addIfService(committerEventHandler);

            //policy handling preemption requests from RM
            callWithJobClassLoader(conf, new Action<Void>() {
                public Void call(Configuration conf) {
                    preemptionPolicy = createPreemptionPolicy(conf);
                    preemptionPolicy.init(context);
                    return null;
                }
            });

            //service to handle requests to TaskUmbilicalProtocol
            // 创建并添加任务监听服务 TaskAttemptListenerImpl
            taskAttemptListener = createTaskAttemptListener(context,
preemptionPolicy);
            addIfService(taskAttemptListener);

            //service to log job history events
            // 创建并注册任务历史事件处理器 JobHistoryEventHandler
            EventHandler<JobHistoryEvent> historyService =
                    createJobHistoryHandler(context);

 dispatcher.register(org.apache.hadoop.mapreduce.jobhistory.EventType.class,
                    historyService);

            // 创建任务事件分发器 JobEventDispatcher
            this.jobEventDispatcher = new JobEventDispatcher();

            //register the event dispatchers
            // 注册相关事件类型
            dispatcher.register(JobEventType.class, jobEventDispatcher);
            dispatcher.register(TaskEventType.class, new TaskEventDispatcher());
            dispatcher.register(TaskAttemptEventType.class,
                    new TaskAttemptEventDispatcher());
            dispatcher.register(CommitterEventType.class, committerEventHandler);

            // 是否开启任务的推测执行
            if (conf.getBoolean(MRJobConfig.MAP_SPECULATIVE, false)
                    || conf.getBoolean(MRJobConfig.REDUCE_SPECULATIVE, false)) {
                //optional service to speculate on task attempts' progress
                // 创建并注册任务推测执行 DefaultSpeculator 服务
                speculator = createSpeculator(conf, context);
                addIfService(speculator);
            }

            // 注册推测执行事件类型
```

```
            speculatorEventDispatcher = new SpeculatorEventDispatcher(conf);
            dispatcher.register(Speculator.EventType.class,
                    speculatorEventDispatcher);

            // Now that there's a FINISHING state for application on RM to give AMs
            // plenty of time to clean up after unregister it's safe to clean
staging
            // directory after unregistering with RM. So, we start the staging-dir
            // cleaner BEFORE the ContainerAllocator so that on shut-down,
            // ContainerAllocator unregisters first and then the staging-dir cleaner
            // deletes staging directory.
            // 创建任务清除目录服务 StagingDirCleaningService
            addService(createStagingDirCleaningService());

            // service to allocate containers from RM (if non-uber) or to fake it
(uber)
            // 添加容器申请器 ContainerAllocatorRouter
            addIfService(containerAllocator);
            dispatcher.register(ContainerAllocator.EventType.class,
containerAllocator);

            // corresponding service to launch allocated containers via NodeManager
            // 创建并注册启动容器服务 ContainerLauncherRouter
            containerLauncher = createContainerLauncher(context);
            addIfService(containerLauncher);
            dispatcher.register(ContainerLauncher.EventType.class,
containerLauncher);

            // Add the JobHistoryEventHandler last so that it is properly stopped
first.
            // This will guarantee that all history-events are flushed before AM
goes
            // ahead with shutdown.
            // Note: Even though JobHistoryEventHandler is started last, if any
            // component creates a JobHistoryEvent in the meanwhile, it will be just
be
            // queued inside the JobHistoryEventHandler
            // 注册任务历史事件处理器 JobHistoryEventHandler
            addIfService(historyService);
        }
        /**
         * MRAppMaster 组合服务的 AsyncDispatcher 异步事件分发器注册了哪些事件注册表
         * 1 EventType               -> JobHistoryEventHandler (任务历史事件处理器)
         * 2 JobEventType            -> JobEventDispatcher (任务事件分发器)
         * 3 TaskEventType           -> TaskEventDispatcher (Task事件分发器)
         * 4 TaskAttemptEventType    -> TaskAttemptEventDispatcher (Task 尝试事件分发
器)
         * 5 CommitterEventType      -> CommitterEventHandler (提交事件处理器)
         * 6 Speculator.EventType    -> SpeculatorEventDispatcher (推测事件处理器)
         * 7 ContainerAllocator.EventType -> ContainerAllocatorRouter (容器申请事件处理
器)
```

```
         * 8 ContainerLauncher.EventType  -> ContainerLauncherRouter (启动容器事件处理
器)
         */

        /**
         * MRAppMaster 组合服务有哪些子服务?
         * 1 异步事件分发器          -> AsyncDispatcher
         * 2 任务完成监控服务        -> TaskAttemptFinishingMonitor
         * 3 提交者事件处理          -> CommitterEventHandler
         * 4 Task监听服务            -> TaskAttemptListenerImpl
         * 5 任务推测执行服务         -> DefaultSpeculator
         * 6 任务清除目录服务         -> StagingDirCleaningService
         * 7 容器申请器服务           -> ContainerAllocatorRouter
         * 8 启动容器服务            -> ContainerLauncherRouter
         * 9 任务历史事件处理器服务   -> JobHistoryEventHandler
         */

        // 调用 MRAppMaster 组合服务的所有子服务 serviceInit()
        super.serviceInit(conf);
    } // end of init()
```

5.2.1.1 异步事件分发器 AsyncDispatcher.serviceInit()

```
// 调用其父类 AbstractService.serviceInit()
```

5.2.1.2 任务完成监控服务 TaskAttemptFinishingMonitor.serviceInit()

```
// 调用其父类 AbstractService.serviceInit()
```

5.2.1.3 提交者事件处理 CommitterEventHandler.serviceInit()

```
public class CommitterEventHandler extends AbstractService
        implements EventHandler<CommitterEvent> {
    @Override
    protected void serviceInit(Configuration conf) throws Exception {
        super.serviceInit(conf);
        // 默认 60s
        commitThreadCancelTimeoutMs = conf.getInt(
                MRJobConfig.MR_AM_COMMITTER_CANCEL_TIMEOUT_MS,
                MRJobConfig.DEFAULT_MR_AM_COMMITTER_CANCEL_TIMEOUT_MS);
        // 默认 10s
        commitWindowMs = conf.getLong(MRJobConfig.MR_AM_COMMIT_WINDOW_MS,
                MRJobConfig.DEFAULT_MR_AM_COMMIT_WINDOW_MS);
        try {
            fs = FileSystem.get(conf);
            JobID id = TypeConverter.fromYarn(context.getApplicationID());
            JobId jobId = TypeConverter.toYarn(id);
            String user = UserGroupInformation.getCurrentUser().getShortUserName();
            startCommitFile = MRApps.getStartJobCommitFile(conf, user, jobId);
            endCommitSuccessFile = MRApps.getEndJobCommitSuccessFile(conf, user,
jobId);
```

```
            endCommitFailureFile = MRApps.getEndJobCommitFailureFile(conf, user,
jobId);
        } catch (IOException e) {
            throw new YarnRuntimeException(e);
        }
    }
}
```

5.2.1.4 Task监听服务 TaskAttemptListenerImpl.serviceInit()

```
/**
 * This class is responsible for talking to the task umblical.
 * It also converts all the old data structures
 * to yarn data structures.
 * <p>
 * This class HAS to be in this package to access package private
 * methods/classes.
 */
public class TaskAttemptListenerImpl extends CompositeService
        implements TaskUmbilicalProtocol, TaskAttemptListener {
     @Override
    protected void serviceInit(Configuration conf) throws Exception {
        // 注册 Task 心跳处理器 (也即 AM 与 Tasks 心跳)
        registerHeartbeatHandler(conf);
        // 默认 10s
        commitWindowMs = conf.getLong(MRJobConfig.MR_AM_COMMIT_WINDOW_MS,
                MRJobConfig.DEFAULT_MR_AM_COMMIT_WINDOW_MS);
        super.serviceInit(conf);
    }
}
```

```
    protected void registerHeartbeatHandler(Configuration conf) {
        // 创建 TaskHeartbeatHandler
        taskHeartbeatHandler = new TaskHeartbeatHandler(context.getEventHandler(),
                context.getClock(),
conf.getInt(MRJobConfig.MR_AM_TASK_LISTENER_THREAD_COUNT,
                MRJobConfig.DEFAULT_MR_AM_TASK_LISTENER_THREAD_COUNT));
        // 添加 TaskHeartbeatHandler 服务
        addService(taskHeartbeatHandler);
    }
```

5.2.1.5 任务推测执行服务 DefaultSpeculator.serviceInit()

```
public class DefaultSpeculator extends AbstractService implements
        Speculator {
 // 调用其父类 AbstractService.serviceInit()
}
```

5.2.1.6 任务清除目录服务 StagingDirCleaningService.serviceInit()

```
private final class StagingDirCleaningService extends AbstractService {
    // 调用其父类 AbstractService.serviceInit()
}
```

5.2.1.7 容器申请器服务 ContainerAllocatorRouter.serviceInit()

```
/**
     * By the time life-cycle of this router starts, job-init would have already
     * happened.
     */
    private final class ContainerAllocatorRouter extends AbstractService
            implements ContainerAllocator, RMHeartbeatHandler {
        // 调用其父类 AbstractService.serviceInit()
    }
```

5.2.1.8 启动容器服务 ContainerLauncherRouter.serviceInit()

```
 /**
     * By the time life-cycle of this router starts, job-init would have already
     * happened.
     */
    private final class ContainerLauncherRouter extends AbstractService
            implements ContainerLauncher {
        // 调用其父类 AbstractService.serviceInit()
    }
```

5.2.2 启动 MRAppMaster (调用 MRAppMaster.serviceStart())

```
@SuppressWarnings("unchecked")
    @Override
    protected void serviceStart() throws Exception {

        /**
         * MRAppMaster 组合服务的 AsyncDispatcher 异步事件分发器注册了哪些事件注册表
         * 1 jobhistory.EventType     -> JobHistoryEventHandler (任务历史事件处理器)
         * 2 JobEventType             -> JobEventDispatcher (任务事件分发器)
         * 3 TaskEventType            -> TaskEventDispatcher (Task事件分发器)
         * 4 TaskAttemptEventType     -> TaskAttemptEventDispatcher (Task 尝试事件分发
器)
         * 5 CommitterEventType       -> CommitterEventHandler (提交事件处理器)
         * 6 Speculator.EventType     -> SpeculatorEventDispatcher (推测事件处理器)
         * 7 ContainerAllocator.EventType -> ContainerAllocatorRouter (容器申请事件处理
器)
         * 8 ContainerLauncher.EventType  -> ContainerLauncherRouter (启动容器事件处理
器)
         */

        amInfos = new LinkedList<AMInfo>();
```

```
        completedTasksFromPreviousRun = new HashMap<TaskId, TaskInfo>();
        processRecovery();
        cleanUpPreviousJobOutput();

        // Current an AMInfo for the current AM generation.
        // 创建当前 AM 信息 AMInfo
        AMInfo amInfo =
                MRBuilderUtils.newAMInfo(appAttemptID, startTime, containerID,
nmHost,
                        nmPort, nmHttpPort);

        // ////////////////// Create the job itself.
        job = createJob(getConfig(), forcedState, shutDownMessage);

        // End of creating the job.

        // Send out an MR AM inited event for all previous AMs.
        // 恢复上一个 AM (默认 amInfos 为空)
        for (AMInfo info : amInfos) {
            dispatcher.getEventHandler().handle(
                    new JobHistoryEvent(job.getID(), new AMStartedEvent(info
                            .getAppAttemptId(), info.getStartTime(),
info.getContainerId(),
                            info.getNodeManagerHost(), info.getNodeManagerPort(),
info
                            .getNodeManagerHttpPort(), appSubmitTime)));
        }

        // Send out an MR AM inited event for this AM.
        // 调用 JobHistoryEventHandler.handle(AMStartedEvent)
        dispatcher.getEventHandler().handle(
                // 创建 JobHistoryEvent
                new JobHistoryEvent(
                        job.getID(),
                        new AMStartedEvent(
                                amInfo.getAppAttemptId(),
                                amInfo.getStartTime(),
                                amInfo.getContainerId(),
                                amInfo.getNodeManagerHost(),
                                amInfo.getNodeManagerPort(), amInfo
                                .getNodeManagerHttpPort(),
                                this.forcedState == null ? null :
this.forcedState.toString(), appSubmitTime)
                ));
        amInfos.add(amInfo);

        // metrics system init is really init & start.
        // It's more test friendly to put it here.
        DefaultMetricsSystem.initialize("MRAppMaster");

        boolean initFailed = false;
        if (!errorHappenedShutDown) {
```

```
            // create a job event for job initialization
            JobEvent initJobEvent = new JobEvent(job.getID(),
JobEventType.JOB_INIT);
            // Send init to the job (this does NOT trigger job execution)
            // This is a synchronous call, not an event through dispatcher. We want
            // job-init to be done completely here.
            // 调用 JobEventDispatcher.handle(JobEventType.JOB_INIT)
            jobEventDispatcher.handle(initJobEvent);

            // If job is still not initialized, an error happened during
            // initialization. Must complete starting all of the services so failure
            // events can be processed.
            initFailed = (((JobImpl) job).getInternalState() !=
JobStateInternal.INITED);

            // JobImpl's InitTransition is done (call above is synchronous), so the
            // "uber-decision" (MR-1220) has been made.  Query job and switch to
            // ubermode if appropriate (by registering different container-allocator
            // and container-launcher services/event-handlers).

            // 默认 false
            if (job.isUber()) {
                speculatorEventDispatcher.disableSpeculation();
                LOG.info("MRAppMaster uberizing job " + job.getID()
                        + " in local container (\"uber-AM\") on node "
                        + nmHost + ":" + nmPort + ".");
            } else {
                // send init to speculator only for non-uber jobs.
                // This won't yet start as dispatcher isn't started yet.
                // 调用
SpeculatorEventDispatcher.handle(Speculator.EventType.JOB_CREATE)
                dispatcher.getEventHandler().handle(
                        new SpeculatorEvent(job.getID(), clock.getTime()));
                // MRAppMaster launching normal, non-uberized, multi-container
                // job job_1684656010852_0006
                LOG.info("MRAppMaster launching normal, non-uberized, multi-
container "
                        + "job " + job.getID() + ".");
            }
            // Start ClientService here, since it's not initialized if
            // errorHappenedShutDown is true
            // 启动 MRClientService RPC 服务 其调用 serviceStart()
            clientService.start();
        }
        //start all the components
        /**
         * MRAppMaster 组合服务有哪些子服务？调用其 serviceStart()
         * 1 异步事件分发器         -> AsyncDispatcher
         * 2 任务完成监控服务       -> TaskAttemptFinishingMonitor
         * 3 提交者事件处理         -> CommitterEventHandler
         * 4 Task监听服务           -> TaskAttemptListenerImpl
         * 5 任务推测执行服务        -> DefaultSpeculator
```

```
         * 6 任务清除目录服务        -> StagingDirCleaningService
         * 7 容器申请器服务          -> ContainerAllocatorRouter
         * 8 启动容器服务           -> ContainerLauncherRouter
         * 9 任务历史事件处理器服务   -> JobHistoryEventHandler
         */
        super.serviceStart();

        // finally set the job classloader
        MRApps.setClassLoader(jobClassLoader, getConfig());

        if (initFailed) {
            JobEvent initFailedEvent = new JobEvent(job.getID(),
JobEventType.JOB_INIT_FAILED);
            jobEventDispatcher.handle(initFailedEvent);
        } else {
            // All components have started, start the job.
            // 全部的组件已经启动 开始启动任务
            startJobs();
        }
    }
```

5.2.2.1 启动 MRClientService RPC 服务 其调用 serviceStart()

```
// 该服务在 RMAppMaster.serviceInit() 创建
// service to handle requests from JobClient
// 创建 MRClientService 服务 该服务是与 JobClient 客户端交互(也即提交任务的客户端)
// JobClient 持续获取提交任务的状态信息
// 通讯协议接口为 MRClientProtocol
/**
 * This module is responsible for talking to the
 * jobclient (user facing).
 */
public class MRClientService extends AbstractService implements ClientService {
 public MRClientService(AppContext appContext) {
        super(MRClientService.class.getName());
        this.appContext = appContext;
        // 创建 MRClientProtocolHandler
        this.protocolHandler = new MRClientProtocolHandler();
    }
    protected void serviceStart() throws Exception {
        Configuration conf = getConfig();
        YarnRPC rpc = YarnRPC.create(conf);
        InetSocketAddress address = new InetSocketAddress(0);

        // 创建 RPC Server 通讯协议为 MRClientProtocol
        // 端口随机
        server =
                rpc.getServer(MRClientProtocol.class, protocolHandler, address,
                        conf, appContext.getClientToAMTokenSecretManager(),
                        conf.getInt(MRJobConfig.MR_AM_JOB_CLIENT_THREAD_COUNT,
                                MRJobConfig.DEFAULT_MR_AM_JOB_CLIENT_THREAD_COUNT),
```

```
                        MRJobConfig.MR_AM_JOB_CLIENT_PORT_RANGE);

        // Enable service authorization?
        if (conf.getBoolean(
                CommonConfigurationKeysPublic.HADOOP_SECURITY_AUTHORIZATION,
                false)) {
            refreshServiceAcls(conf, new MRAMPolicyProvider());
        }

        // 启动 RPC 服务
        server.start();
        this.bindAddress =
NetUtils.createSocketAddrForHost(appContext.getNMHostname(),
                server.getListenerAddress().getPort());
        // Instantiated MRClientService at hadoop103/192.168.6.103:37605
        LOG.info("Instantiated MRClientService at " + this.bindAddress);
        try {
            // Explicitly disabling SSL for map reduce task as we can't allow MR
users
            // to gain access to keystore file for opening SSL listener. We can
trust
            // RM/NM to issue SSL certificates but definitely not MR-AM as it is
            // running in user-land.
            // 创建并启动 WebApp
            webApp =
                    WebApps.$for("mapreduce", AppContext.class, appContext, "ws")
                            .withHttpPolicy(conf, Policy.HTTP_ONLY)
                            .withPortRange(conf,
MRJobConfig.MR_AM_WEBAPP_PORT_RANGE)
                            .start(new AMWebApp());
        } catch (Exception e) {
            LOG.error("Webapps failed to start. Ignoring for now:", e);
        }
        super.serviceStart();
    }
}
```

5.2.2.2 异步事件分发器 AsyncDispatcher.serviceStart()

```
    @Override
    protected void serviceStart() throws Exception {
        //start all the components
        // 针对 ResourceManager 的 AsyncDispatcher 服务来说 啥也不干
        super.serviceStart();
        // 创建 EventHandler 线程
        eventHandlingThread = new Thread(createThread());
        eventHandlingThread.setName(dispatcherThreadName);
        eventHandlingThread.start();
    }
```

5.2.2.3 任务完成监控服务 TaskAttemptFinishingMonitor.serviceStart()

```
// 调用其父类 AbstractLivelinessMonitor.serviceStart()
  @Override
  protected void serviceStart() throws Exception {
    assert !stopped : "starting when already stopped";
    // 重置 Timer
    resetTimer();
    // 创建并启动 PingChecker 线程
    checkerThread = new Thread(new PingChecker());
    checkerThread.setName("Ping Checker");
    checkerThread.start();
    super.serviceStart();
  }
```

5.2.2.4 提交者事件处理 CommitterEventHandler.serviceStart()

```
@Override
    protected void serviceStart() throws Exception {
        ThreadFactoryBuilder tfBuilder = new ThreadFactoryBuilder()
                .setNameFormat("CommitterEvent Processor #%d");
        // 默认为 null
        if (jobClassLoader != null) {
            // if the job classloader is enabled, we need to use the job classloader
            // as the thread context classloader (TCCL) of these threads in case the
            // committer needs to load another class via TCCL
            ThreadFactory backingTf = new ThreadFactory() {
                @Override
                public Thread newThread(Runnable r) {
                    Thread thread = new Thread(r);
                    thread.setContextClassLoader(jobClassLoader);
                    return thread;
                }
            };
            tfBuilder.setThreadFactory(backingTf);
        }
        ThreadFactory tf = tfBuilder.build();
        // 创建线程池
        launcherPool = new HadoopThreadPoolExecutor(5, 5, 1,
                TimeUnit.HOURS, new LinkedBlockingQueue<Runnable>(), tf);
        // 创建并启动线程
        eventHandlingThread = new Thread(new Runnable() {
            @Override
            public void run() {
                CommitterEvent event = null;
                while (!stopped.get() && !Thread.currentThread().isInterrupted()) {
                    try {
                        event = eventQueue.take();
                    } catch (InterruptedException e) {
                        if (!stopped.get()) {
                            LOG.error("Returning, interrupted : " + e);
```

```
                        }
                        return;
                    }
                    // the events from the queue are handled in parallel
                    // using a thread pool
                    // 执行
                    launcherPool.execute(new EventProcessor(event));
                }
            }
        });
        eventHandlingThread.setName("CommitterEvent Handler");
        eventHandlingThread.start();
        super.serviceStart();
    }
```

5.2.2.5 Task监听服务 TaskAttemptListenerImpl.serviceStart()

```
    @Override
    protected void serviceStart() throws Exception {
        // 启动 RPC Server
        startRpcServer();
        super.serviceStart();
    }
```

```
protected void startRpcServer() {
        Configuration conf = getConfig();
        try {
            // 构建 RPC Server 通讯协议接口为 TaskUmbilicalProtocol 端口随机
            server = new RPC.Builder(conf).setProtocol(TaskUmbilicalProtocol.class)
                    .setInstance(this).setBindAddress("0.0.0.0")
                    .setPortRangeConfig(MRJobConfig.MR_AM_JOB_CLIENT_PORT_RANGE)
                    .setNumHandlers(

 conf.getInt(MRJobConfig.MR_AM_TASK_LISTENER_THREAD_COUNT,

 MRJobConfig.DEFAULT_MR_AM_TASK_LISTENER_THREAD_COUNT))

.setVerbose(false).setSecretManager(jobTokenSecretManager).build();

            // Enable service authorization?
            if (conf.getBoolean(
                    CommonConfigurationKeysPublic.HADOOP_SECURITY_AUTHORIZATION,
                    false)) {
                refreshServiceAcls(conf, new MRAMPolicyProvider());
            }

            // 启动 RPC 服务
            server.start();

            // 解析 RPC Server 的 HOST PORT
            this.address = NetUtils.createSocketAddrForHost(
```

```
                    context.getNMHostname(),
                    server.getListenerAddress().getPort());
        } catch (IOException e) {
            throw new YarnRuntimeException(e);
        }
    }
```

5.2.2.6 任务推测执行服务 DefaultSpeculator.serviceStart()

```
@Override
    protected void serviceStart() throws Exception {
        // 创建并运行推测执行后台线程
        Runnable speculationBackgroundCore
                = new Runnable() {
            @Override
            public void run() {
                while (!stopped && !Thread.currentThread().isInterrupted()) {
                    long backgroundRunStartTime = clock.getTime();
                    try {
                        int speculations = computeSpeculations();
                        long mininumRecomp
                                = speculations > 0 ? soonestRetryAfterSpeculate
                                : soonestRetryAfterNoSpeculate;

                        long wait = Math.max(mininumRecomp,
                                clock.getTime() - backgroundRunStartTime);

                        if (speculations > 0) {
                            LOG.info("We launched " + speculations
                                    + " speculations.  Sleeping " + wait + "
milliseconds.");
                        }

                        Object pollResult
                                = scanControl.poll(wait, TimeUnit.MILLISECONDS);
                    } catch (InterruptedException e) {
                        if (!stopped) {
                            LOG.error("Background thread returning, interrupted",
e);
                        }
                        return;
                    }
                }
            }
        };
        speculationBackgroundThread = new Thread
                (speculationBackgroundCore, "DefaultSpeculator background
processing");
        speculationBackgroundThread.start();

        super.serviceStart();
```

5.2.2.7 任务清除目录服务 StagingDirCleaningService.serviceStart()

```
// 调用其父类 AbstractService.serviceStart()
```

5.2.2.8 容器申请器服务 ContainerAllocatorRouter.serviceStart()

```
@Override
        protected void serviceStart() throws Exception {
            if (job.isUber()) {
                MRApps.setupDistributedCacheLocal(getConfig());
                this.containerAllocator = new LocalContainerAllocator(
                        this.clientService, this.context, nmHost, nmPort, nmHttpPort
                        , containerID);
            } else {
                // 创建从 RM 申请容器服务 RMContainerAllocator
                this.containerAllocator = new RMContainerAllocator(
                        this.clientService, this.context, preemptionPolicy);
            }
            // 调用 RMContainerAllocator.serviceInit()
            ((Service) this.containerAllocator).init(getConfig());
            // 调用 RMContainerAllocator.serviceStart()
            ((Service) this.containerAllocator).start();
            super.serviceStart();
        }
```

5.2.2.8.1 创建从 RM 申请容器服务 RMContainerAllocator

```
/**
 * Allocates the container from the ResourceManager scheduler.
 */
public class RMContainerAllocator extends RMContainerRequestor
        implements ContainerAllocator {
     public RMContainerAllocator(ClientService clientService, AppContext context,
                                 AMPreemptionPolicy preemptionPolicy) {
        super(clientService, context);
        this.preemptionPolicy = preemptionPolicy;
        this.stopped = new AtomicBoolean(false);
        this.clock = context.getClock();
        // 创建 AssignedRequests
        this.assignedRequests = createAssignedRequests();
    }
}
```

```
/**
 * Keeps the data structures to send container requests to RM.
 */
public abstract class RMContainerRequestor extends RMCommunicator {
   public RMContainerRequestor(ClientService clientService, AppContext context) {
    super(clientService, context);
  }
}
```

```
/**
 * Registers/unregisters to RM and sends heartbeats to RM.
 */
public abstract class RMCommunicator extends AbstractService
    implements RMHeartbeatHandler {
   public RMCommunicator(ClientService clientService, AppContext context) {
    super("RMCommunicator");
    this.clientService = clientService;
    this.context = context;
    this.eventHandler = context.getEventHandler();
    this.applicationId = context.getApplicationID();
    this.stopped = new AtomicBoolean(false);
    this.heartbeatCallbacks = new ConcurrentLinkedQueue<Runnable>();
    this.schedulerResourceTypes = EnumSet.of(SchedulerResourceTypes.MEMORY);
  }
}
```

5.2.2.8.2 初始化调用 RMContainerAllocator.serviceInit()

```
@Override
    protected void serviceInit(Configuration conf) throws Exception {
        // 调用父类
        super.serviceInit(conf);
        reduceSlowStart = conf.getFloat(
                MRJobConfig.COMPLETED_MAPS_FOR_REDUCE_SLOWSTART,
                DEFAULT_COMPLETED_MAPS_PERCENT_FOR_REDUCE_SLOWSTART);
        maxReduceRampupLimit = conf.getFloat(
                MRJobConfig.MR_AM_JOB_REDUCE_RAMPUP_UP_LIMIT,
                MRJobConfig.DEFAULT_MR_AM_JOB_REDUCE_RAMP_UP_LIMIT);
        maxReducePreemptionLimit = conf.getFloat(
                MRJobConfig.MR_AM_JOB_REDUCE_PREEMPTION_LIMIT,
                MRJobConfig.DEFAULT_MR_AM_JOB_REDUCE_PREEMPTION_LIMIT);
        reducerUnconditionalPreemptionDelayMs = 1000 * conf.getInt(
                MRJobConfig.MR_JOB_REDUCER_UNCONDITIONAL_PREEMPT_DELAY_SEC,
                MRJobConfig.DEFAULT_MR_JOB_REDUCER_UNCONDITIONAL_PREEMPT_DELAY_SEC);
        reducerNoHeadroomPreemptionDelayMs = conf.getInt(
                MRJobConfig.MR_JOB_REDUCER_PREEMPT_DELAY_SEC,
                MRJobConfig.DEFAULT_MR_JOB_REDUCER_PREEMPT_DELAY_SEC) * 1000;//sec -
> ms
        maxRunningMaps = conf.getInt(MRJobConfig.JOB_RUNNING_MAP_LIMIT,
                MRJobConfig.DEFAULT_JOB_RUNNING_MAP_LIMIT);
```

```
        maxRunningReduces = conf.getInt(MRJobConfig.JOB_RUNNING_REDUCE_LIMIT,
                MRJobConfig.DEFAULT_JOB_RUNNING_REDUCE_LIMIT);
        RackResolver.init(conf);
        retryInterval =
getConfig().getLong(MRJobConfig.MR_AM_TO_RM_WAIT_INTERVAL_MS,
                MRJobConfig.DEFAULT_MR_AM_TO_RM_WAIT_INTERVAL_MS);
        mapNodeLabelExpression = conf.get(MRJobConfig.MAP_NODE_LABEL_EXP);
        reduceNodeLabelExpression = conf.get(MRJobConfig.REDUCE_NODE_LABEL_EXP);
        // Init startTime to current time. If all goes well, it will be reset after
        // first attempt to contact RM.
        retrystartTime = System.currentTimeMillis();
        this.scheduledRequests.setNumOpportunisticMapsPercent(
                conf.getInt(MRJobConfig.MR_NUM_OPPORTUNISTIC_MAPS_PERCENT,
                        MRJobConfig.DEFAULT_MR_NUM_OPPORTUNISTIC_MAPS_PERCENT));
        //  0% of the mappers will be scheduled using OPPORTUNISTIC containers
        LOG.info(this.scheduledRequests.getNumOpportunisticMapsPercent() +
                "% of the mappers will be scheduled using OPPORTUNISTIC
containers");
    }
```

```
@Override
    protected void serviceInit(Configuration conf) throws Exception {
        // 调用父类
        super.serviceInit(conf);
        nodeBlacklistingEnabled =
                conf.getBoolean(MRJobConfig.MR_AM_JOB_NODE_BLACKLISTING_ENABLE,
true);
        // nodeBlacklistingEnabled:true
        LOG.info("nodeBlacklistingEnabled:" + nodeBlacklistingEnabled);
        maxTaskFailuresPerNode =
                conf.getInt(MRJobConfig.MAX_TASK_FAILURES_PER_TRACKER, 3);
        blacklistDisablePercent =
                conf.getInt(

 MRJobConfig.MR_AM_IGNORE_BLACKLISTING_BLACKLISTED_NODE_PERECENT,

 MRJobConfig.DEFAULT_MR_AM_IGNORE_BLACKLISTING_BLACKLISTED_NODE_PERCENT);
        // maxTaskFailuresPerNode is 3
        LOG.info("maxTaskFailuresPerNode is " + maxTaskFailuresPerNode);
        if (blacklistDisablePercent < -1 || blacklistDisablePercent > 100) {
            throw new YarnRuntimeException("Invalid blacklistDisablePercent: "
                    + blacklistDisablePercent
                    + ". Should be an integer between 0 and 100 or -1 to disabled");
        }
        //  blacklistDisablePercent is 33
        LOG.info("blacklistDisablePercent is " + blacklistDisablePercent);
    }
```

5.2.2.8.3 启动调用 RMContainerAllocator.serviceStart()

```
@Override
    protected void serviceStart() throws Exception {
        // 创建线程并启动
        this.eventHandlingThread = new Thread() {
            @SuppressWarnings("unchecked")
            @Override
            public void run() {

                ContainerAllocatorEvent event;

                while (!stopped.get() && !Thread.currentThread().isInterrupted()) {
                    try {
                        // 拉取 event
                        event = RMContainerAllocator.this.eventQueue.take();
                    } catch (InterruptedException e) {
                        if (!stopped.get()) {
                            LOG.error("Returning, interrupted : " + e);
                        }
                        return;
                    }

                    try {
                        // 处理 event
                        handleEvent(event);
                    } catch (Throwable t) {
                        LOG.error("Error in handling event type " + event.getType()
                                + " to the ContainreAllocator", t);
                        // Kill the AM
                        eventHandler.handle(new JobEvent(getJob().getID(),
                                JobEventType.INTERNAL_ERROR));
                        return;
                    }
                }
            }
        };
        this.eventHandlingThread.start();
        // 调用父类
        super.serviceStart();
    }
```

```
@Override
    protected void serviceStart() throws Exception {
        // 创建 ResourceManager 的 ApplicationMasterService RPC 服务的客户端代理对象
        // 通讯协议接口为 ApplicationMasterProtocol
        scheduler = createSchedulerProxy();
        JobID id = TypeConverter.fromYarn(this.applicationId);
        JobId jobId = TypeConverter.toYarn(id);
        job = context.getJob(jobId);
        // 向 ResourceManager 的 ApplicationMasterService RPC 服务注册 AM
```

```
        register();
        // 启动容器申请线程 AllocatorRunnable 也即发送心跳
        startAllocatorThread();
        super.serviceStart();
    }
```

5.2.2.9 启动容器服务 ContainerLauncherRouter.serviceStart()

```
 @Override
        protected void serviceStart() throws Exception {
            if (job.isUber()) {
                this.containerLauncher = new LocalContainerLauncher(context,
                        (TaskUmbilicalProtocol) taskAttemptListener,
jobClassLoader);
                ((LocalContainerLauncher) this.containerLauncher)
                        .setEncryptedSpillKey(encryptedSpillKey);
            } else {
                // 创建启动容器服务 ContainerLauncherImpl
                this.containerLauncher = new ContainerLauncherImpl(context);
            }
            // 调用 ContainerLauncherImpl.serviceInit()
            ((Service) this.containerLauncher).init(getConfig());
            // 调用 ContainerLauncherImpl.serviceStart()
            ((Service) this.containerLauncher).start();
            // 调用父类
            super.serviceStart();
        }
```

5.2.2.9.1 创建启动容器服务 ContainerLauncherImpl

```
/**
 * This class is responsible for launching of containers.
 */
public class ContainerLauncherImpl extends AbstractService implements
        ContainerLauncher {
     public ContainerLauncherImpl(AppContext context) {
        super(ContainerLauncherImpl.class.getName());
        this.context = context;
        this.stopped = new AtomicBoolean(false);
    }
}
```

5.2.2.9.2 初始化调用 ContainerLauncherImpl.serviceInit()

```
@Override
    protected void serviceInit(Configuration conf) throws Exception {
        // 默认 500
        this.limitOnPoolSize = conf.getInt(
                MRJobConfig.MR_AM_CONTAINERLAUNCHER_THREAD_COUNT_LIMIT,
                MRJobConfig.DEFAULT_MR_AM_CONTAINERLAUNCHER_THREAD_COUNT_LIMIT);
        // Upper limit on the thread pool size is 500
```

```
        LOG.info("Upper limit on the thread pool size is " + this.limitOnPoolSize);

        this.initialPoolSize = conf.getInt(
                MRJobConfig.MR_AM_CONTAINERLAUNCHER_THREADPOOL_INITIAL_SIZE,

 MRJobConfig.DEFAULT_MR_AM_CONTAINERLAUNCHER_THREADPOOL_INITIAL_SIZE);
        // The thread pool initial size is 10
        LOG.info("The thread pool initial size is " + this.initialPoolSize);

        super.serviceInit(conf);
        // 创建 ContainerManagementProtocolProxy
        cmProxy = new ContainerManagementProtocolProxy(conf);
    }
```

```
/**
 * Helper class to manage container manager proxies
 */
@LimitedPrivate({"MapReduce", "YARN"})
public class ContainerManagementProtocolProxy {
 public ContainerManagementProtocolProxy(Configuration conf,
                                         NMTokenCache nmTokenCache) {
        this.conf = new Configuration(conf);
        this.nmTokenCache = nmTokenCache;

        // 默认 0
        maxConnectedNMs =
                conf.getInt(YarnConfiguration.NM_CLIENT_MAX_NM_PROXIES,
                        YarnConfiguration.DEFAULT_NM_CLIENT_MAX_NM_PROXIES);
        if (maxConnectedNMs < 0) {
            throw new YarnRuntimeException(
                    YarnConfiguration.NM_CLIENT_MAX_NM_PROXIES
                            + " (" + maxConnectedNMs + ") can not be less than 0.");
        }

        if (LOG.isDebugEnabled()) {
            LOG.debug(YarnConfiguration.NM_CLIENT_MAX_NM_PROXIES + " : " +
                    maxConnectedNMs);
        }

        if (maxConnectedNMs > 0) {
            cmProxy =
                    new LinkedHashMap<String, ContainerManagementProtocolProxyData>
();
        } else {
            cmProxy = Collections.emptyMap();
            // Connections are not being cached so ensure connections close quickly
            // to avoid creating thousands of RPC client threads on large clusters.
            this.conf.setInt(

 CommonConfigurationKeysPublic.IPC_CLIENT_CONNECTION_MAXIDLETIME_KEY,
                    0);
```

```
        }
        // 创建 HadoopYarnProtoRPC
        rpc = YarnRPC.create(conf);
    }
}
```

5.2.2.9.3 启动调用 ContainerLauncherImpl.serviceStart()

```
protected void serviceStart() throws Exception {

        ThreadFactory tf = new ThreadFactoryBuilder().setNameFormat(
                "ContainerLauncher #%d").setDaemon(true).build();

        // Start with a default core-pool size of 10 and change it dynamically.
        launcherPool = new HadoopThreadPoolExecutor(initialPoolSize,
                Integer.MAX_VALUE, 1, TimeUnit.HOURS,
                new LinkedBlockingQueue<Runnable>(),
                tf);
        // 创建线程并启动
        eventHandlingThread = new Thread() {
            @Override
            public void run() {
                ContainerLauncherEvent event = null;
                Set<String> allNodes = new HashSet<String>();

                while (!stopped.get() && !Thread.currentThread().isInterrupted()) {
                    try {
                        // 阻塞拉取 event
                        event = eventQueue.take();
                    } catch (InterruptedException e) {
                        if (!stopped.get()) {
                            LOG.error("Returning, interrupted : " + e);
                        }
                        return;
                    }
                    allNodes.add(event.getContainerMgrAddress());

                    int poolSize = launcherPool.getCorePoolSize();

                    // See if we need up the pool size only if haven't reached the
                    // maximum limit yet.
                    if (poolSize != limitOnPoolSize) {

                        // nodes where containers will run at *this* point of time.
This is
                        // *not* the cluster size and doesn't need to be.
                        int numNodes = allNodes.size();
                        int idealPoolSize = Math.min(limitOnPoolSize, numNodes);

                        if (poolSize < idealPoolSize) {
                            // Bump up the pool size to
idealPoolSize+initialPoolSize, the
```

```
                            // later is just a buffer so we are not always
increasing the
                            // pool-size
                            int newPoolSize = Math.min(limitOnPoolSize,
idealPoolSize
                                    + initialPoolSize);
                            LOG.info("Setting ContainerLauncher pool size to " +
newPoolSize
                                    + " as number-of-nodes to talk to is " +
numNodes);
                            launcherPool.setCorePoolSize(newPoolSize);
                        }
                    }

                    // the events from the queue are handled in parallel
                    // using a thread pool
                    launcherPool.execute(createEventProcessor(event));

                    // TODO: Group launching of multiple containers to a single
                    // NodeManager into a single connection
                }
            }
        };
        eventHandlingThread.setName("ContainerLauncher Event Handler");
        eventHandlingThread.start();
        super.serviceStart();
    }
```

5.4 AM 的所有组件基本启动完成开始执行任务

```
// All components have started, start the job.
// 全部的组件已经启动 开始启动任务
startJobs();
```

```
/**
     * This can be overridden to instantiate multiple jobs and create a
     * workflow.
     * <p>
     * TODO:  Rework the design to actually support this.  Currently much of the
     * job stuff has been moved to init() above to support uberization (MR-1220).
     * In a typical workflow, one presumably would want to uberize only a subset
     * of the jobs (the "small" ones), which is awkward with the current design.
     */
    @SuppressWarnings("unchecked")
    protected void startJobs() {
        /** create a job-start event to get this ball rolling */
        // 创建 job-start 事件
        // eventType = JobEventType.JOB_START
        JobEvent startJobEvent = new JobStartEvent(job.getID(),
                recoveredJobStartTime);
```

```
        /** send the job-start event. this triggers the job execution. */
        // 调用 JobEventDispatcher.handle(JobEventType.JOB_START)
        dispatcher.getEventHandler().handle(startJobEvent);
    }
```

5.4.1 启动 MR 任务创建待执行 MapTask、ReduceTask 的源码分析

```
// 在启动 MRAppMaster.serviceStart() 时 触发了一个 JobEventType.JOB_INIT
// 核心源码入口如下：
if (!errorHappenedShutDown) {
            // create a job event for job initialization
            // 初始化 Job (也即创建 MapTask ReduceTask 等待后续被执行调度)
            JobEvent initJobEvent = new JobEvent(job.getID(),
JobEventType.JOB_INIT);
            // Send init to the job (this does NOT trigger job execution)
            // This is a synchronous call, not an event through dispatcher. We want
            // job-init to be done completely here.
            // 调用 JobEventDispatcher.handle(JobEventType.JOB_INIT)
            jobEventDispatcher.handle(initJobEvent);
.....}
```

```
// 经过各种状态转换 最终来到 InitTransition.doTransition(JobEventType.JOB_INIT)
public static class InitTransition
            implements MultipleArcTransition<JobImpl, JobEvent, JobStateInternal> {

        /**
         * Note that this transition method is called directly (and synchronously)
         * by MRAppMaster's init() method (i.e., no RPC, no thread-switching;
         * just plain sequential call within AM context), so we can trigger
         * modifications in AM state from here (at least, if AM is written that
         * way; MR version is).
         */
        @Override
        public JobStateInternal transition(JobImpl job, JobEvent event) {
            // evenType = JobEventType.JOB_INIT
            job.metrics.submittedJob(job);
            job.metrics.preparingJob(job);

            if (job.newApiCommitter) {
                // 创建 JobImpl 上下文 JobContextImpl
                job.jobContext = new JobContextImpl(job.conf,
                        job.oldJobId);
            } else {
                job.jobContext = new org.apache.hadoop.mapred.JobContextImpl(
                        job.conf, job.oldJobId);
            }

            try {
                setup(job);
                job.fs = job.getFileSystem(job.conf);
```

```
                // log to job history
                JobSubmittedEvent jse = new JobSubmittedEvent(job.oldJobId,
                        job.conf.get(MRJobConfig.JOB_NAME, "test"),
                        job.conf.get(MRJobConfig.USER_NAME, "mapred"),
                        job.appSubmitTime,
                        job.remoteJobConfFile.toString(),
                        job.jobACLs, job.queueName,
                        job.conf.get(MRJobConfig.WORKFLOW_ID, ""),
                        job.conf.get(MRJobConfig.WORKFLOW_NAME, ""),
                        job.conf.get(MRJobConfig.WORKFLOW_NODE_NAME, ""),
                        getWorkflowAdjacencies(job.conf),
                        job.conf.get(MRJobConfig.WORKFLOW_TAGS, ""), job.conf);
                job.eventHandler.handle(new JobHistoryEvent(job.jobId, jse));
                //TODO JH Verify jobACLs, UserName via UGI?

                // 获取 MR 任务切片元数据信息 (也即读取 HDFS 的切片信息)
                TaskSplitMetaInfo[] taskSplitMetaInfo = createSplits(job,
job.jobId);
                // 切片有多少个 MapTask 就有几个
                job.numMapTasks = taskSplitMetaInfo.length;
                // 默认 ReduceTask 为 0
                job.numReduceTasks = job.conf.getInt(MRJobConfig.NUM_REDUCES, 0);

                if (job.numMapTasks == 0 && job.numReduceTasks == 0) {
                    job.addDiagnostic("No of maps and reduces are 0 " + job.jobId);
                } else if (job.numMapTasks == 0) {
                    job.reduceWeight = 0.9f;
                } else if (job.numReduceTasks == 0) {
                    job.mapWeight = 0.9f;
                } else {
                    job.mapWeight = job.reduceWeight = 0.45f;
                }

                checkTaskLimits();

                long inputLength = 0;
                for (int i = 0; i < job.numMapTasks; ++i) {
                    inputLength += taskSplitMetaInfo[i].getInputDataLength();
                }

                job.makeUberDecision(inputLength);

                job.taskAttemptCompletionEvents =
                        new ArrayList<TaskAttemptCompletionEvent>(
                                job.numMapTasks + job.numReduceTasks + 10);
                job.mapAttemptCompletionEvents =
                        new ArrayList<TaskCompletionEvent>(job.numMapTasks + 10);
                job.taskCompletionIdxToMapCompletionIdx = new ArrayList<Integer>(
                        job.numMapTasks + job.numReduceTasks + 10);

                job.allowedMapFailuresPercent =
```

```
                        job.conf.getInt(MRJobConfig.MAP_FAILURES_MAX_PERCENT, 0);
                job.allowedReduceFailuresPercent =
                        job.conf.getInt(MRJobConfig.REDUCE_FAILURES_MAXPERCENT, 0);

                cleanupSharedCacheUploadPolicies(job.conf);

                // create the Tasks but don't start them yet
                // 创建 MapTask、ReduceTask 任务 (重要、重要、重要)
                createMapTasks(job, inputLength, taskSplitMetaInfo);
                createReduceTasks(job);

                job.metrics.endPreparingJob(job);
                // 返回 JobStateInternal.INITED
                return JobStateInternal.INITED;
            } catch (Exception e) {
                LOG.warn("Job init failed", e);
                job.metrics.endPreparingJob(job);
                job.addDiagnostic("Job init failed : "
                        + StringUtils.stringifyException(e));
                // Leave job in the NEW state. The MR AM will detect that the state
is
                // not INITED and send a JOB_INIT_FAILED event.
                return JobStateInternal.NEW;
            }
        }
```

```
// 创建 MapTaskImpl 并缓存
private void createMapTasks(JobImpl job, long inputLength,
                                TaskSplitMetaInfo[] splits) {
            for (int i = 0; i < job.numMapTasks; ++i) {
                // 创建 MapTaskImpl (也即 MapTask)
                TaskImpl task =
                        new MapTaskImpl(job.jobId, i,
                                job.eventHandler,
                                job.remoteJobConfFile,
                                job.conf, splits[i],
                                job.taskAttemptListener,
                                job.jobToken, job.jobCredentials,
                                job.clock,
                                job.applicationAttemptId.getAttemptId(),
                                job.metrics, job.appContext);
                // 添加待执行 MapTask
                job.addTask(task);
            }
            //  Input size for job job_1684656010852_0006 = 147145. Number of splits
= 1
            LOG.info("Input size for job " + job.jobId + " = " + inputLength
                    + ". Number of splits = " + splits.length);
        }
```

```
// 创建 ReduceTaskImpl 并缓存
```

```
private void createReduceTasks(JobImpl job) {
            for (int i = 0; i < job.numReduceTasks; i++) {
                TaskImpl task =
                        new ReduceTaskImpl(job.jobId, i,
                                job.eventHandler,
                                job.remoteJobConfFile,
                                job.conf, job.numMapTasks,
                                job.taskAttemptListener, job.jobToken,
                                job.jobCredentials, job.clock,
                                job.applicationAttemptId.getAttemptId(),
                                job.metrics, job.appContext);
                job.addTask(task);
            }
            // Number of reduces for job job_1684656010852_0006 = 1
            LOG.info("Number of reduces for job " + job.jobId + " = "
                    + job.numReduceTasks);
        }
```

5.4.2 开始执行任务经过多种状态转换最终调用 SetupCompletedTransition.doTransition() 执行 MapTask或者 ReduceTask

```
private static class SetupCompletedTransition
            implements SingleArcTransition<JobImpl, JobEvent> {
        @Override
        public void transition(JobImpl job, JobEvent event) {
            // eventType = JobEventType.JOB_SETUP_COMPLETED
            job.setupProgress = 1.0f;
            // 调度执行 MapTask (缓存 MapTasks 在 JobImpl.addTask())
            job.scheduleTasks(job.mapTasks, job.numReduceTasks == 0);
            // 调度执行 ReduceTask (缓存 ReduceTasks 在 JobImpl.addTask())
            job.scheduleTasks(job.reduceTasks, true);

            // If we have no tasks, just transition to job completed
            // 如果没有 Task 则表示任务执行完成
            if (job.numReduceTasks == 0 && job.numMapTasks == 0) {
                // 调用 JobEventDispatcher.handle(JobEventType.JOB_COMPLETED)
                job.eventHandler.handle(new JobEvent(job.jobId,
                        JobEventType.JOB_COMPLETED));
            }
        }
    }
```

```
protected void scheduleTasks(Set<TaskId> taskIDs,
                                boolean recoverTaskOutput) {
        for (TaskId taskID : taskIDs) {
            TaskInfo taskInfo = completedTasksFromPreviousRun.remove(taskID);
            if (taskInfo != null) {
                eventHandler.handle(new TaskRecoverEvent(taskID, taskInfo,
                        committer, recoverTaskOutput));
            } else {
```

```
                // 处理启动 MapTask 任务
                // 处理启动 ReduceTask 任务
                // 调用 TaskEventDispatcher.handle(TaskEventType.T_SCHEDULE))
                eventHandler.handle(new TaskEvent(taskID,
TaskEventType.T_SCHEDULE));
            }
        }
    }
```

5.4.3 通过各种状态转换之后向 RM 的 ContainerManagerImpl RPC 服务申请容器启动MapTask或者ReduceTask源码分析

```
// 最终调用 RMContainerAllocator.handle(ContainerAllocator.EventType.CONTAINER_REQ) 处
理容器申请
@Override
    public void handle(ContainerAllocatorEvent event) {
        // eventType = ContainerAllocator.EventType.CONTAINER_REQ
        int qSize = eventQueue.size();
        if (qSize != 0 && qSize % 1000 == 0) {
            LOG.info("Size of event-queue in RMContainerAllocator is " + qSize);
        }
        int remCapacity = eventQueue.remainingCapacity();
        if (remCapacity < 1000) {
            LOG.warn("Very low remaining capacity in the event-queue "
                    + "of RMContainerAllocator: " + remCapacity);
        }
        try {
            // 添加申请容器事件
            // 调用 serviceStart() 启动线程的 run()
            eventQueue.put(event);
        } catch (InterruptedException e) {
            throw new YarnRuntimeException(e);
        }
    }
```

```
 @Override
    protected void serviceStart() throws Exception {
        // 创建线程并启动
        this.eventHandlingThread = new Thread() {
            @SuppressWarnings("unchecked")
            @Override
            public void run() {

                ContainerAllocatorEvent event;
                // eventType =

                while (!stopped.get() && !Thread.currentThread().isInterrupted()) {
                    try {
                        // 拉取 event (申请容器启动 MapTask 或者 ReduceTask)
                        event = RMContainerAllocator.this.eventQueue.take();
```

```
                    } catch (InterruptedException e) {
                        if (!stopped.get()) {
                            LOG.error("Returning, interrupted : " + e);
                        }
                        return;
                    }

                    try {
                        // 处理申请容器 event
                        handleEvent(event);
                    } catch (Throwable t) {
                        LOG.error("Error in handling event type " + event.getType()
                                + " to the ContainreAllocator", t);
                        // Kill the AM
                        eventHandler.handle(new JobEvent(getJob().getID(),
                                JobEventType.INTERNAL_ERROR));
                        return;
                    }
                }
            }
        };
        this.eventHandlingThread.start();
        // 调用父类
        super.serviceStart();
    }
```

```
protected synchronized void handleEvent(ContainerAllocatorEvent event) {
        recalculateReduceSchedule = true;
        // evenType = ContainerAllocator.EventType.CONTAINER_REQ
        if (event.getType() == ContainerAllocator.EventType.CONTAINER_REQ) {
            ContainerRequestEvent reqEvent = (ContainerRequestEvent) event;
            boolean isMap = reqEvent.getAttemptID().getTaskId().getTaskType().
                    equals(TaskType.MAP);
            if (isMap) {
                // 处理申请启动 MapTask 容器请求
                handleMapContainerRequest(reqEvent);
            } else {
                // 处理申请启动 ReduceTask 容器请求
                handleReduceContainerRequest(reqEvent);
            }

        } else if (
                event.getType() ==
ContainerAllocator.EventType.CONTAINER_DEALLOCATE) {

            LOG.info("Processing the event " + event.toString());

            TaskAttemptId aId = event.getAttemptID();

            boolean removed = scheduledRequests.remove(aId);
            if (!removed) {
```

```
                ContainerId containerId = assignedRequests.get(aId);
                if (containerId != null) {
                    removed = true;
                    assignedRequests.remove(aId);
                    containersReleased++;
                    pendingRelease.add(containerId);
                    release(containerId);
                }
            }
            if (!removed) {
                LOG.error("Could not deallocate container for task attemptId " +
                        aId);
            }
            preemptionPolicy.handleCompletedContainer(event.getAttemptID());
        } else if (
                event.getType() == ContainerAllocator.EventType.CONTAINER_FAILED) {
            ContainerFailedEvent fEv = (ContainerFailedEvent) event;
            String host = getHost(fEv.getContMgrAddress());
            containerFailedOnHost(host);
            // propagate failures to preemption policy to discard checkpoints for
            // failed tasks
            preemptionPolicy.handleFailedContainer(event.getAttemptID());
        }
    }
```

5.4.3.1 处理 MapTask 容器请求

```
@SuppressWarnings({"unchecked"})
    private void handleMapContainerRequest(ContainerRequestEvent reqEvent) {
        assert (reqEvent.getAttemptID().getTaskId().getTaskType().equals(
                TaskType.MAP));

        // 获取容器可以申请的最大资源配置
        Resource supportedMaxContainerCapability = getMaxContainerCapability();
        JobId jobId = getJob().getID();

        if (mapResourceRequest.equals(Resources.none())) {
            mapResourceRequest = reqEvent.getCapability();
            eventHandler.handle(new JobHistoryEvent(jobId,
                    new NormalizedResourceEvent(
                            org.apache.hadoop.mapreduce.TaskType.MAP,
                            mapResourceRequest.getMemorySize())));
            // 默认容器申请资源 <memory:1024, vCores:1>
            LOG.info("mapResourceRequest:" + mapResourceRequest);
        }

        boolean mapContainerRequestAccepted = true;
        // 判断启动 MapTask 容器的资源是否大于容器的最大申请资源
        if (mapResourceRequest.getMemorySize() >
                supportedMaxContainerCapability.getMemorySize()
                ||
                mapResourceRequest.getVirtualCores() >
```

```
                        supportedMaxContainerCapability.getVirtualCores()) {
            mapContainerRequestAccepted = false;
        }

        // mapContainerRequestAccepted = true 表示申请启动 MapTask 容器
        // 的资源没有大于容器的最大申请资源
        // 默认容器申请资源为 <memory:1024, vCores:1>
        if (mapContainerRequestAccepted) {
            // set the resources
            // 申请容器请求设置资源
            reqEvent.getCapability().setMemorySize(
                    mapResourceRequest.getMemorySize());
            reqEvent.getCapability().setVirtualCores(
                    mapResourceRequest.getVirtualCores());
            // 添加请求
            scheduledRequests.addMap(reqEvent); //maps are immediately scheduled
        } else {
            String diagMsg = "The required MAP capability is more than the " +
                    "supported max container capability in the cluster. Killing" +
                    " the Job. mapResourceRequest: " + mapResourceRequest +
                    " maxContainerCapability:" + supportedMaxContainerCapability;
            LOG.info(diagMsg);
            eventHandler.handle(new JobDiagnosticsUpdateEvent(jobId, diagMsg));
            eventHandler.handle(new JobEvent(jobId, JobEventType.JOB_KILL));
        }
    }
```

```
void addMap(ContainerRequestEvent event) {
            ContainerRequest request = null;

            if (event.getEarlierAttemptFailed()) {
                earlierFailedMaps.add(event.getAttemptID());
                request =
                        new ContainerRequest(event, PRIORITY_FAST_FAIL_MAP,
                                mapNodeLabelExpression);
                LOG.info("Added " + event.getAttemptID() + " to list of failed
maps");
                // If its an earlier Failed attempt, do not retry as OPPORTUNISTIC
                maps.put(event.getAttemptID(), request);
                addContainerReq(request);
            } else {
                if (mapsMod100 < numOpportunisticMapsPercent) {
                    request =
                            new ContainerRequest(event, PRIORITY_OPPORTUNISTIC_MAP,
                                    mapNodeLabelExpression);
                    maps.put(event.getAttemptID(), request);
                    addOpportunisticResourceRequest(request.priority,
request.capability);
                } else {
                    // 创建容器请求 ContainerRequest
                    request =
```

```
                            new ContainerRequest(event, PRIORITY_MAP,
mapNodeLabelExpression);
                    // 申请容器优先级 本地优先
                    for (String host : event.getHosts()) {
                        LinkedList<TaskAttemptId> list = mapsHostMapping.get(host);
                        if (list == null) {
                            list = new LinkedList<TaskAttemptId>();
                            mapsHostMapping.put(host, list);
                        }
                        list.add(event.getAttemptID());
                        if (LOG.isDebugEnabled()) {
                            LOG.debug("Added attempt req to host " + host);
                        }
                    }
                    // 申请容器优先级 其次机架
                    for (String rack : event.getRacks()) {
                        LinkedList<TaskAttemptId> list = mapsRackMapping.get(rack);
                        if (list == null) {
                            list = new LinkedList<TaskAttemptId>();
                            mapsRackMapping.put(rack, list);
                        }
                        list.add(event.getAttemptID());
                        if (LOG.isDebugEnabled()) {
                            LOG.debug("Added attempt req to rack " + rack);
                        }
                    }
                    // 缓存 AttemptID -> ContainerRequest
                    maps.put(event.getAttemptID(), request);
                    // 添加申请容器请求
                    addContainerReq(request);
                }
                mapsMod100++;
                mapsMod100 %= 100;
            }
        }
```

```
protected void addContainerReq(ContainerRequest req) {
        // Create resource requests
        for (String host : req.hosts) {
            // Data-local
            // 数据本地性 (判断启动 Task 容器的 host 不在 NM 黑名单)
            if (!isNodeBlacklisted(host)) {
                addResourceRequest(req.priority, host, req.capability,
                        null);
            }
        }

        // Nothing Rack-local for now
        // 机架
        for (String rack : req.racks) {
            addResourceRequest(req.priority, rack, req.capability,
```

```
                    null);
        }

        // Off-switch
        addResourceRequest(req.priority, ResourceRequest.ANY, req.capability,
                req.nodeLabelExpression);
    }
```

```
    private void addResourceRequest(Priority priority, String resourceName,
                                    Resource capability, String nodeLabelExpression)
{
        // 添加资源(容器)申请
        addResourceRequest(priority, resourceName, capability, nodeLabelExpression,
                ExecutionType.GUARANTEED);
    }
```

```
private void addResourceRequest(Priority priority, String resourceName,
                                    Resource capability, String nodeLabelExpression,
                                    ExecutionType executionType) {
        Map<String, Map<Resource, ResourceRequest>> remoteRequests =
                this.remoteRequestsTable.get(priority);
        if (remoteRequests == null) {
            remoteRequests = new HashMap<String, Map<Resource, ResourceRequest>>();
            this.remoteRequestsTable.put(priority, remoteRequests);
            if (LOG.isDebugEnabled()) {
                LOG.debug("Added priority=" + priority);
            }
        }
        Map<Resource, ResourceRequest> reqMap = remoteRequests.get(resourceName);
        if (reqMap == null) {
            reqMap = new HashMap<Resource, ResourceRequest>();
            remoteRequests.put(resourceName, reqMap);
        }
        ResourceRequest remoteRequest = reqMap.get(capability);
        if (remoteRequest == null) {
            remoteRequest = recordFactory.newRecordInstance(ResourceRequest.class);
            remoteRequest.setPriority(priority);
            remoteRequest.setResourceName(resourceName);
            remoteRequest.setCapability(capability);
            remoteRequest.setNumContainers(0);
            remoteRequest.setNodeLabelExpression(nodeLabelExpression);
            remoteRequest.setExecutionTypeRequest(
                    ExecutionTypeRequest.newInstance(executionType, true));
            reqMap.put(capability, remoteRequest);
        }
        remoteRequest.setNumContainers(remoteRequest.getNumContainers() + 1);

        // Note this down for next interaction with ResourceManager
        // 申请容器请求 (将 ResourceRequest 请求缓存等待其他线程拉取)
        // 调用 AllocatorRunnable.run() 发送心跳给 RM 的 ApplicationMasterService 申请容
器
```

```
        // 最终调用 RMContainerRequestor.makeRemoteRequest()
        addResourceRequestToAsk(remoteRequest);
        if (LOG.isDebugEnabled()) {
            LOG.debug("addResourceRequest:" + " applicationId="
                    + applicationId.getId() + " priority=" + priority.getPriority()
                    + " resourceName=" + resourceName + " numContainers="
                    + remoteRequest.getNumContainers() + " #asks=" + ask.size());
        }
    }
```

```
    private void addResourceRequestToAsk(ResourceRequest remoteRequest) {
        // because objects inside the resource map can be deleted ask can end up
        // containing an object that matches new resource object but with different
        // numContainers. So existing values must be replaced explicitly
        ask.remove(remoteRequest);
        ask.add(remoteRequest);
    }
```

5.4.3.1.1 AM 申请容器启动 MapTask 容器是通过发送心跳向 RM 的 ContainerManagerImpl RPC 服务申请资源(调用ApplicationMasterService.allocate())

```
// 调用 AllocatorRunnable.run() 发送心跳给 RM 的 ApplicationMasterService 申请容器
@VisibleForTesting
    public class AllocatorRunnable implements Runnable {
        @Override
        public void run() {
            while (!stopped.get() && !Thread.currentThread().isInterrupted()) {
                try {
                    // 默认睡眠 1s
                    Thread.sleep(rmPollInterval);
                    try {
                        // 心跳 调用 RMContainerAllocator.heartbeat()
                        heartbeat();
                    } catch (RMContainerAllocationException e) {
                        LOG.error("Error communicating with RM: " + e.getMessage(),
e);

                        return;
                    } catch (Exception e) {
                        LOG.error("ERROR IN CONTACTING RM. ", e);
                        continue;
                        // TODO: for other exceptions
                    }

                    lastHeartbeatTime = context.getClock().getTime();

                    // 执行心跳回调
                    executeHeartbeatCallbacks();
                } catch (InterruptedException e) {
                    if (!stopped.get()) {
                        LOG.warn("Allocated thread interrupted. Returning.");
                    }
```

```
                    return;
                }
            }
        }
    }
```

```
@Override
    protected synchronized void heartbeat() throws Exception {
        scheduleStats.updateAndLogIfChanged("Before Scheduling: ");
        // 获取 AM 基于分配的容器 (里面发送 RPC 请求申请启动 Task 容器)
        // 也即 allocatedContainers 是已经申请好的容器资源
        List<Container> allocatedContainers = getResources();
        if (allocatedContainers != null && allocatedContainers.size() > 0) {
            // 分配容器启动 Task
            scheduledRequests.assign(allocatedContainers);
        }

        // 获取已完成 MAP 任务数
        int completedMaps = getJob().getCompletedMaps();
        // 获取已经完成任务数
        int completedTasks = completedMaps + getJob().getCompletedReduces();

        if ((lastCompletedTasks != completedTasks) ||
                (scheduledRequests.maps.size() > 0)) {
            lastCompletedTasks = completedTasks;
            recalculateReduceSchedule = true;
        }

        if (recalculateReduceSchedule) {
            boolean reducerPreempted = preemptReducesIfNeeded();

            if (!reducerPreempted) {
                // Only schedule new reducers if no reducer preemption happens for
                // this heartbeat
                scheduleReduces(getJob().getTotalMaps(), completedMaps,
                        scheduledRequests.maps.size(),
scheduledRequests.reduces.size(),
                        assignedRequests.maps.size(),
assignedRequests.reduces.size(),
                        mapResourceRequest, reduceResourceRequest,
pendingReduces.size(),
                        maxReduceRampupLimit, reduceSlowStart);
            }

            recalculateReduceSchedule = false;
        }

        scheduleStats.updateAndLogIfChanged("After Scheduling: ");
    }
```

```
@SuppressWarnings("unchecked")
```

```
    private List<Container> getResources() throws Exception {
        applyConcurrentTaskLimits();

        // will be null the first time
        Resource headRoom = Resources.clone(getAvailableResources());
        AllocateResponse response;
        /*
         * If contact with RM is lost, the AM will wait MR_AM_TO_RM_WAIT_INTERVAL_MS
         * milliseconds before aborting. During this interval, AM will still try
         * to contact the RM.
         */
        try {
            // 发送 RPC 请求申请容器并返回
            response = makeRemoteRequest();
            // Reset retry count if no exception occurred.
            retrystartTime = System.currentTimeMillis();
        } catch (ApplicationAttemptNotFoundException e) {
            // This can happen if the RM has been restarted. If it is in that state,
            // this application must clean itself up.
            eventHandler.handle(new JobEvent(this.getJob().getID(),
                    JobEventType.JOB_AM_REBOOT));
            throw new RMContainerAllocationException(
                    "Resource Manager doesn't recognize AttemptId: "
                            + this.getContext().getApplicationAttemptId(), e);
        } catch (ApplicationMasterNotRegisteredException e) {
            LOG.info("ApplicationMaster is out of sync with ResourceManager,"
                    + " hence resync and send outstanding requests.");
            // RM may have restarted, re-register with RM.
            lastResponseID = 0;
            register();
            addOutstandingRequestOnResync();
            return null;
        } catch (InvalidLabelResourceRequestException e) {
            // If Invalid label exception is received means the requested label
doesnt
            // have access so killing job in this case.
            String diagMsg = "Requested node-label-expression is invalid: "
                    + StringUtils.stringifyException(e);
            LOG.info(diagMsg);
            JobId jobId = this.getJob().getID();
            eventHandler.handle(new JobDiagnosticsUpdateEvent(jobId, diagMsg));
            eventHandler.handle(new JobEvent(jobId, JobEventType.JOB_KILL));
            throw e;
        } catch (Exception e) {
            // This can happen when the connection to the RM has gone down. Keep
            // re-trying until the retryInterval has expired.
            if (System.currentTimeMillis() - retrystartTime >= retryInterval) {
                LOG.error("Could not contact RM after " + retryInterval + "
milliseconds.");
                eventHandler.handle(new JobEvent(this.getJob().getID(),
                        JobEventType.JOB_AM_REBOOT));
```

```
                throw new RMContainerAllocationException("Could not contact RM after
" +
                        retryInterval + " milliseconds.");
            }
            // Throw this up to the caller, which may decide to ignore it and
            // continue to attempt to contact the RM.
            throw e;
        }
        Resource newHeadRoom = getAvailableResources();
        // 获取申请容器集合
        List<Container> newContainers = response.getAllocatedContainers();
        // Setting NMTokens
        if (response.getNMTokens() != null) {
            for (NMToken nmToken : response.getNMTokens()) {
                NMTokenCache.setNMToken(nmToken.getNodeId().toString(),
                        nmToken.getToken());
            }
        }

        // Setting AMRMToken
        if (response.getAMRMToken() != null) {
            updateAMRMToken(response.getAMRMToken());
        }

        List<ContainerStatus> finishedContainers =
                response.getCompletedContainersStatuses();

        // propagate preemption requests
        final PreemptionMessage preemptReq = response.getPreemptionMessage();
        if (preemptReq != null) {
            preemptionPolicy.preempt(
                    new PreemptionContext(assignedRequests), preemptReq);
        }

        if (newContainers.size() + finishedContainers.size() > 0
                || !headRoom.equals(newHeadRoom)) {
            //something changed
            recalculateReduceSchedule = true;
            if (LOG.isDebugEnabled() && !headRoom.equals(newHeadRoom)) {
                LOG.debug("headroom=" + newHeadRoom);
            }
        }

        if (LOG.isDebugEnabled()) {
            for (Container cont : newContainers) {
                LOG.debug("Received new Container :" + cont);
            }
        }

        //Called on each allocation. Will know about newly blacklisted/added hosts.
        computeIgnoreBlacklisting();
```

```
        handleUpdatedNodes(response);
        handleJobPriorityChange(response);
        // Handle receiving the timeline collector address and token for this app.
        MRAppMaster.RunningAppContext appContext =
                (MRAppMaster.RunningAppContext) this.getContext();
        if (appContext.getTimelineV2Client() != null) {
            appContext.getTimelineV2Client().
                    setTimelineCollectorInfo(response.getCollectorInfo());
        }
        for (ContainerStatus cont : finishedContainers) {
            processFinishedContainer(cont);
        }
        // 返回申请好的容器资源
        return newContainers;
    }
```

```
protected AllocateResponse makeRemoteRequest() throws YarnException,
            IOException {
        // 请求容器资源限制检查
        applyRequestLimits();
        // 申请容器的 NM 黑名单
        ResourceBlacklistRequest blacklistRequest =
                ResourceBlacklistRequest.newInstance(new ArrayList<String>
(blacklistAdditions),
                        new ArrayList<String>(blacklistRemovals));
        // 创建申请容器请求 AllocateRequest
        AllocateRequest allocateRequest =
                AllocateRequest.newInstance(lastResponseID,
                        super.getApplicationProgress(),
                        // ask 就是申请目标容器的集合
                        new ArrayList<ResourceRequest>(ask),
                        new ArrayList<ContainerId>(release), blacklistRequest);
        // 发送 RPC 请求申请容器 (调用 ApplicationMasterService.allocate())
        // 返回申请容器结果
        AllocateResponse allocateResponse = scheduler.allocate(allocateRequest);
        lastResponseID = allocateResponse.getResponseId();
        availableResources = allocateResponse.getAvailableResources();
        lastClusterNmCount = clusterNmCount;
        clusterNmCount = allocateResponse.getNumClusterNodes();
        int numCompletedContainers =
                allocateResponse.getCompletedContainersStatuses().size();

        if (ask.size() > 0 || release.size() > 0) {
            /**
             * getResources() for application_1684656010852_0006:
             * ask=3 release= 0
             * newContainers=0 finishedContainers=0
             * resourcelimit=<memory:10752, vCores:23> knownNMs=3
             */
            LOG.info("getResources() for " + applicationId + ":" + " ask="
                    + ask.size() + " release= " + release.size() + " newContainers="
```

```
                    + allocateResponse.getAllocatedContainers().size()
                    + " finishedContainers=" + numCompletedContainers
                    + " resourcelimit=" + availableResources + " knownNMs="
                    + clusterNmCount);
        }

        ask.clear();
        release.clear();

        if (numCompletedContainers > 0) {
            // re-send limited requests when a container completes to trigger asking
            // for more containers
            requestLimitsToUpdate.addAll(requestLimits.keySet());
        }

        if (blacklistAdditions.size() > 0 || blacklistRemovals.size() > 0) {
            LOG.info("Update the blacklist for " + applicationId +
                    ": blacklistAdditions=" + blacklistAdditions.size() +
                    " blacklistRemovals=" + blacklistRemovals.size());
        }
        blacklistAdditions.clear();
        blacklistRemovals.clear();
        return allocateResponse;
    }
```

5.4.3.1.2 容器申请好之后开始做状态准备以及分配容器

```
// 分配启动容器 调用 ScheduledRequests.assign(allocatedContainers)
// this method will change the list of allocatedContainers.
        private void assign(List<Container> allocatedContainers) {
            // allocatedContainers 是已经申请好的容器资源
            Iterator<Container> it = allocatedContainers.iterator();
            // Got allocated containers 1
            LOG.info("Got allocated containers " + allocatedContainers.size());
            containersAllocated += allocatedContainers.size();
            int reducePending = reduces.size();

            // 校验申请好的容器信息是否合法 (比如启动容器的 NM 信息是否在黑名单中)
            while (it.hasNext()) {
                Container allocated = it.next();
                if (LOG.isDebugEnabled()) {
                    LOG.debug("Assigning container " + allocated.getId()
                            + " with priority " + allocated.getPriority() + " to NM
"
                            + allocated.getNodeId());
                }

                // check if allocated container meets memory requirements
                // and whether we have any scheduled tasks that need
                // a container to be assigned
                boolean isAssignable = true;
                Priority priority = allocated.getPriority();
```

```
                // 获取申请好的容器资源
                Resource allocatedResource = allocated.getResource();

                if (PRIORITY_FAST_FAIL_MAP.equals(priority)
                        || PRIORITY_MAP.equals(priority)
                        || PRIORITY_OPPORTUNISTIC_MAP.equals(priority)) {
                    if
(ResourceCalculatorUtils.computeAvailableContainers(allocatedResource,
                            mapResourceRequest, getSchedulerResourceTypes()) <= 0
                            || maps.isEmpty()) {
                        LOG.info("Cannot assign container " + allocated
                                + " for a map as either "
                                + " container memory less than required " +
mapResourceRequest
                                + " or no pending map tasks - maps.isEmpty="
                                + maps.isEmpty());
                        isAssignable = false;
                    }
                } else if (PRIORITY_REDUCE.equals(priority)) {
                    if
(ResourceCalculatorUtils.computeAvailableContainers(allocatedResource,
                            reduceResourceRequest, getSchedulerResourceTypes()) <= 0
                            || (reducePending <= 0)) {
                        LOG.info("Cannot assign container " + allocated
                                + " for a reduce as either "
                                + " container memory less than required " +
reduceResourceRequest
                                + " or no pending reduce tasks.");
                        isAssignable = false;
                    } else {
                        reducePending--;
                    }
                } else {
                    LOG.warn("Container allocated at unwanted priority: " + priority
+
                            ". Returning to RM...");
                    isAssignable = false;
                }

                if (!isAssignable) {
                    // release container if we could not assign it
                    containerNotAssigned(allocated);
                    it.remove();
                    continue;
                }

                // do not assign if allocated container is on a
                // blacklisted host
                // 获取启动容器在 NM 的主机信息
                String allocatedHost = allocated.getNodeId().getHost();
                // 如果启动容器在 NM 的主机信息是在 NM 的黑名单 一般情况都不在黑名单
                if (isNodeBlacklisted(allocatedHost)) {
```

```
                    // we need to request for a new container
                    // and release the current one
                    LOG.info("Got allocated container on a blacklisted "
                            + " host " + allocatedHost
                            + ". Releasing container " + allocated);

                    // find the request matching this allocated container
                    // and replace it with a new one
                    ContainerRequest toBeReplacedReq =
                            getContainerReqToReplace(allocated);
                    if (toBeReplacedReq != null) {
                        LOG.info("Placing a new container request for task attempt "
                                + toBeReplacedReq.attemptID);
                        ContainerRequest newReq =
                                getFilteredContainerRequest(toBeReplacedReq);
                        decContainerReq(toBeReplacedReq);
                        if (toBeReplacedReq.attemptID.getTaskId().getTaskType() ==
                                TaskType.MAP) {
                            maps.put(newReq.attemptID, newReq);
                        } else {
                            reduces.put(newReq.attemptID, newReq);
                        }
                        addContainerReq(newReq);
                    } else {
                        LOG.info("Could not map allocated container to a valid
request."
                                + " Releasing allocated container " + allocated);
                    }

                    // release container if we could not assign it
                    containerNotAssigned(allocated);
                    it.remove();
                    continue;
                }
            }

            // 到这里说明申请好的容器 (也即启动容器在 NM) 是合法可用
            // 接下来就是根据容器的 NM 信息去连接 NM 启动对应的 Task
            assignContainers(allocatedContainers);

            // release container if we could not assign it
            // 一般情况下 allocatedContainers 在 assignContainers() 已经使用移除啦
            // 所以 it.hasNext() = false
            it = allocatedContainers.iterator();
            while (it.hasNext()) {
                Container allocated = it.next();
                LOG.info("Releasing unassigned container " + allocated);
                containerNotAssigned(allocated);
            }
        }
```

```
private void assignContainers(List<Container> allocatedContainers) {
            // 分配容器
            Iterator<Container> it = allocatedContainers.iterator();
            while (it.hasNext()) {
                // 分配好的容器 (向 RM 申请返回的)
                Container allocated = it.next();
                // 尝试分配容器 正常情况下是分配不成功的 (看里面的逻辑)
                ContainerRequest assigned = assignWithoutLocality(allocated);
                if (assigned != null) {
                    // 分配容器 (正常情况下)
                    containerAssigned(allocated, assigned);
                    it.remove();
                }
            }

            // 正常情况下 到这里基本完成容器的分配 也即 allocatedContainers 为空
            // 这种情况是数据本地性 也即数据在当前节点 那么启动容器也在当前节点 避免数据的网络传输
            assignMapsWithLocality(allocatedContainers);
        }
```

```
 @SuppressWarnings("unchecked")
        private void containerAssigned(Container allocated,
                                       ContainerRequest assigned) {
            // Update resource requests
            // 更新容器资源
            decContainerReq(assigned);

            // send the container-assigned event to task attempt
            // 调用
TaskAttemptEventDispatcher.handle(TaskAttemptEventType.TA_ASSIGNED)
            eventHandler.handle(new TaskAttemptContainerAssignedEvent(
                    assigned.attemptID, allocated, applicationACLs));

            assignedRequests.add(allocated, assigned.attemptID);

            if (LOG.isDebugEnabled()) {
                LOG.debug("Assigned container (" + allocated + ") "
                        + " to task " + assigned.attemptID + " on node "
                        + allocated.getNodeId().toString());
            }
        }
```

5.4.3.1.3 分配容器的状态转换之后最终调用 ContainerAssignedTransition.doTransition(TaskAttemptEventType.TA_ASSIGNED) 构建启动容器的进程(YarnChild)上下文环境

```
private static class ContainerAssignedTransition implements
            SingleArcTransition<TaskAttemptImpl, TaskAttemptEvent> {
        @SuppressWarnings({"unchecked"})
        @Override
        public void transition(final TaskAttemptImpl taskAttempt,
```

```
                                TaskAttemptEvent event) {
            // eventType = TaskAttemptEventType.TA_ASSIGNED
            final TaskAttemptContainerAssignedEvent cEvent =
                    (TaskAttemptContainerAssignedEvent) event;
            // 获取申请好的容器启动 Task
            Container container = cEvent.getContainer();
            taskAttempt.container = container;
            // this is a _real_ Task (classic Hadoop mapred flavor):
            taskAttempt.remoteTask = taskAttempt.createRemoteTask();
            taskAttempt.jvmID =
                    new WrappedJvmID(taskAttempt.remoteTask.getTaskID().getJobID(),
                            taskAttempt.remoteTask.isMapTask(),
                            taskAttempt.container.getId().getContainerId());
            taskAttempt.taskAttemptListener.registerPendingTask(
                    taskAttempt.remoteTask, taskAttempt.jvmID);

            taskAttempt.computeRackAndLocality();

            //launch the container
            //create the container object to be launched for a given Task attempt
            // 创建启动容器上下文的 ContainerLaunchContext
            // 也即容器启动 YarnChild 进程相关命令以及环境变量
            ContainerLaunchContext launchContext = createContainerLaunchContext(
                    cEvent.getApplicationACLs(), taskAttempt.conf,
taskAttempt.jobToken,
                    taskAttempt.remoteTask, taskAttempt.oldJobId, taskAttempt.jvmID,
                    taskAttempt.taskAttemptListener, taskAttempt.credentials);
            // 调用
ContainerLauncherRouter.handle(ContainerLauncher.EventType.CONTAINER_REMOTE_LAUNCH)
            taskAttempt.eventHandler
                    .handle(new ContainerRemoteLaunchEvent(taskAttempt.attemptId,
                            launchContext, container, taskAttempt.remoteTask));

            // send event to speculator that our container needs are satisfied
            // 发送事件给推测执行器 让其分析启动容器
            // 调用
SpeculatorEventDispatcher.handle(Speculator.EventType.TASK_CONTAINER_NEED_UPDATE)
            taskAttempt.eventHandler.handle
                    (new SpeculatorEvent(taskAttempt.getID().getTaskId(), -1));
        }
    }
```

```
static ContainerLaunchContext createContainerLaunchContext(
            Map<ApplicationAccessType, String> applicationACLs,
            Configuration conf, Token<JobTokenIdentifier> jobToken, Task remoteTask,
            final org.apache.hadoop.mapred.JobID oldJobId,
            WrappedJvmID jvmID,
            TaskAttemptListener taskAttemptListener,
            Credentials credentials) {

        synchronized (commonContainerSpecLock) {
```

```
            if (commonContainerSpec == null) {
                // 构建通用的容器启动进程模板 此时还没有对应启动进程命令
                commonContainerSpec = createCommonContainerLaunchContext(
                        applicationACLs, conf, jobToken, oldJobId, credentials);
            }
        }

        // Fill in the fields needed per-container that are missing in the common
        // spec.

        boolean userClassesTakesPrecedence =
                conf.getBoolean(MRJobConfig.MAPREDUCE_JOB_USER_CLASSPATH_FIRST,
false);

        // Setup environment by cloning from common env.
        Map<String, String> env = commonContainerSpec.getEnvironment();
        Map<String, String> myEnv = new HashMap<String, String>(env.size());
        myEnv.putAll(env);
        if (userClassesTakesPrecedence) {
            myEnv.put(Environment.CLASSPATH_PREPEND_DISTCACHE.name(), "true");
        }
        MapReduceChildJVM.setVMEnv(myEnv, remoteTask);

        // Set up the launch command
        // 构建启动容器进行命令 (也即启动容器啥进程 也即 YarnChild 进程)
        List<String> commands = MapReduceChildJVM.getVMCommand(
                taskAttemptListener.getAddress(), remoteTask, jvmID);

        // Duplicate the ByteBuffers for access by multiple containers.
        Map<String, ByteBuffer> myServiceData = new HashMap<String, ByteBuffer>();
        for (Entry<String, ByteBuffer> entry : commonContainerSpec
                .getServiceData().entrySet()) {
            myServiceData.put(entry.getKey(), entry.getValue().duplicate());
        }

        // Construct the actual Container
        // 构建容器上下文对象 也即准备好了容器启动 YarnChild 进程的所有准备工作
        ContainerLaunchContext container = ContainerLaunchContext.newInstance(
                commonContainerSpec.getLocalResources(), myEnv, commands,
                myServiceData, commonContainerSpec.getTokens().duplicate(),
                applicationACLs);

        return container;
    }
```

```
public class MapReduceChildJVM {
 public static List<String> getVMCommand(
      InetSocketAddress taskAttemptListenerAddr, Task task,
      JVMId jvmID) {

    TaskAttemptID attemptID = task.getTaskID();
```

```
    JobConf conf = task.conf;

    Vector<String> vargs = new Vector<String>(8);

    vargs.add(MRApps.crossPlatformifyMREnv(task.conf, Environment.JAVA_HOME)
        + "/bin/java");

    // Add child (task) java-vm options.
    //
    // The following symbols if present in mapred.{map|reduce}.child.java.opts
    // value are replaced:
    // + @taskid@ is interpolated with value of TaskID.
    // Other occurrences of @ will not be altered.
    //
    // Example with multiple arguments and substitutions, showing
    // jvm GC logging, and start of a passwordless JVM JMX agent so can
    // connect with jconsole and the likes to watch child memory, threads
    // and get thread dumps.
    //
    //  <property>
    //    <name>mapred.map.child.java.opts</name>
    //    <value>-Xmx 512M -verbose:gc -Xloggc:/tmp/@taskid@.gc \
    //           -Dcom.sun.management.jmxremote.authenticate=false \
    //           -Dcom.sun.management.jmxremote.ssl=false \
    //    </value>
    //  </property>
    //
    //  <property>
    //    <name>mapred.reduce.child.java.opts</name>
    //    <value>-Xmx 1024M -verbose:gc -Xloggc:/tmp/@taskid@.gc \
    //           -Dcom.sun.management.jmxremote.authenticate=false \
    //           -Dcom.sun.management.jmxremote.ssl=false \
    //    </value>
    //  </property>
    //
    String javaOpts = getChildJavaOpts(conf, task.isMapTask());
    javaOpts = javaOpts.replace("@taskid@", attemptID.toString());
    String [] javaOptsSplit = javaOpts.split(" ");
    for (int i = 0; i < javaOptsSplit.length; i++) {
      vargs.add(javaOptsSplit[i]);
    }

    Path childTmpDir = new Path(MRApps.crossPlatformifyMREnv(conf, Environment.PWD),
        YarnConfiguration.DEFAULT_CONTAINER_TEMP_DIR);
    vargs.add("-Djava.io.tmpdir=" + childTmpDir);
    MRApps.addLog4jSystemProperties(task, vargs, conf);

    if (conf.getProfileEnabled()) {
      if (conf.getProfileTaskRange(task.isMapTask()
                                   ).isIncluded(task.getPartition())) {
        final String profileParams = conf.get(task.isMapTask()
            ? MRJobConfig.TASK_MAP_PROFILE_PARAMS
```

```
            : MRJobConfig.TASK_REDUCE_PROFILE_PARAMS, conf.getProfileParams());
        vargs.add(String.format(profileParams,
            getTaskLogFile(TaskLog.LogName.PROFILE)));
      }
    }

    // Add main class and its arguments
    vargs.add(YarnChild.class.getName());  // main of Child
    // pass TaskAttemptListener's address
    vargs.add(taskAttemptListenerAddr.getAddress().getHostAddress());
    vargs.add(Integer.toString(taskAttemptListenerAddr.getPort()));
    vargs.add(attemptID.toString());                      // pass task identifier

    // Finally add the jvmID
    vargs.add(String.valueOf(jvmID.getId()));
    vargs.add("1>" + getTaskLogFile(TaskLog.LogName.STDOUT));
    vargs.add("2>" + getTaskLogFile(TaskLog.LogName.STDERR));

    // Final commmand
    StringBuilder mergedCommand = new StringBuilder();
    for (CharSequence str : vargs) {
      mergedCommand.append(str).append(" ");
    }
    Vector<String> vargsFinal = new Vector<String>(1);
    vargsFinal.add(mergedCommand.toString());
    return vargsFinal;
  }
}
```

5.4.3.1.4 启动容器进程(YarnChild)相关环境准备好之后连接 NM 的 ContainerManagerImpl 服务 发送 RPC 请求启动容器 YarnChild 进程(调用 ContainerManagerImpl.startContainers() 类似启动 MRAppMaster 进程)

```
// 最终调用 ContainerLauncherImpl.EventProcessor.run()
/**
     * Setup and start the container on remote nodemanager.
     */
    class EventProcessor implements Runnable {
        private ContainerLauncherEvent event;

        EventProcessor(ContainerLauncherEvent event) {
            // eventType = ContainerLauncher.EventType.CONTAINER_REMOTE_LAUNCH
            this.event = event;
        }

        @Override
        public void run() {
            // Processing the event
            // EventType: CONTAINER_REMOTE_LAUNCH
            // for container container_1684656010852_0006_01_000002
            // taskAttempt attempt_1684656010852_0006_m_000000_0
            LOG.info("Processing the event " + event.toString());
```

```
            // Load ContainerManager tokens before creating a connection.
            // TODO: Do it only once per NodeManager.
            ContainerId containerID = event.getContainerID();

            Container c = getContainer(event);
            switch (event.getType()) {

                case CONTAINER_REMOTE_LAUNCH:
                    ContainerRemoteLaunchEvent launchEvent
                            = (ContainerRemoteLaunchEvent) event;
                    // 部署容器
                    c.launch(launchEvent);
                    break;

                case CONTAINER_REMOTE_CLEANUP:
                    c.kill(event.getDumpContainerThreads());
                    break;

                case CONTAINER_COMPLETED:
                    c.done();
                    break;

            }
            removeContainerIfDone(containerID);
        }
    }
```

```
// 调用 ContainerLauncherImpl.Container.launch() 发送 RPC 请求部署容器
 @SuppressWarnings("unchecked")
        public synchronized void launch(ContainerRemoteLaunchEvent event) {
            // Launching attempt_1684656010852_0006_m_000000_0
            LOG.info("Launching " + taskAttemptID);
            if (this.state == ContainerState.KILLED_BEFORE_LAUNCH) {
                state = ContainerState.DONE;
                sendContainerLaunchFailedMsg(taskAttemptID,
                        "Container was killed before it was launched");
                return;
            }

            ContainerManagementProtocolProxyData proxy = null;
            try {
                // 获取 NM 的 RPC ContainerManagerImpl 服务客户端代理
                // 通讯协议接口为 ContainerManagementProtocol
                proxy = getCMProxy(containerMgrAddress, containerID);

                // Construct the actual Container
                // 获取部署容器上下文对象 ContainerLaunchContext
                ContainerLaunchContext containerLaunchContext =
                        event.getContainerLaunchContext();
```

```
                // Now launch the actual container
                // 创建启动容器 RPC 请求体
                StartContainerRequest startRequest =
                        StartContainerRequest.newInstance(containerLaunchContext,
                                event.getContainerToken());
                List<StartContainerRequest> list = new
ArrayList<StartContainerRequest>();
                list.add(startRequest);
                StartContainersRequest requestList =
StartContainersRequest.newInstance(list);

                // 发送 RPC 请求启动容器
                // 调用 ContainerManagerImpl.startContainers()
                StartContainersResponse response =
                        proxy.getContainerManagementProtocol()
                                .startContainers(requestList);

                if (response.getFailedRequests() != null
                        && response.getFailedRequests().containsKey(containerID)) {
                    throw
response.getFailedRequests().get(containerID).deSerialize();
                }
                ByteBuffer portInfo =
                        response.getAllServicesMetaData().get(
                                ShuffleHandler.MAPREDUCE_SHUFFLE_SERVICEID);
                int port = -1;
                if (portInfo != null) {
                    port = ShuffleHandler.deserializeMetaData(portInfo);
                }
                LOG.info("Shuffle port returned by ContainerManager for "
                        + taskAttemptID + " : " + port);

                if (port < 0) {
                    this.state = ContainerState.FAILED;
                    throw new IllegalStateException("Invalid shuffle port number "
                            + port + " returned for " + taskAttemptID);
                }

                // after launching, send launched event to task attempt to move
                // it from ASSIGNED to RUNNING state
                context.getEventHandler().handle(
                        new TaskAttemptContainerLaunchedEvent(taskAttemptID, port));
                this.state = ContainerState.RUNNING;
            } catch (Throwable t) {
                String message = "Container launch failed for " + containerID + " :
"
                        + StringUtils.stringifyException(t);
                this.state = ContainerState.FAILED;
                sendContainerLaunchFailedMsg(taskAttemptID, message);
            } finally {
                if (proxy != null) {
                    cmProxy.mayBeCloseProxy(proxy);
```

```
                }
            }
        }
```

5.4.3.2 处理 ReduceTask 容器请求(类似启动 MapTask)

六 启动 MR 真正处理任务 YarnChild 进程

```
/**
 * The main() for MapReduce task processes.
 */
class YarnChild {
 public static void main(String[] args) throws Throwable {
        Thread.setDefaultUncaughtExceptionHandler(new
YarnUncaughtExceptionHandler());
        LOG.debug("Child starting");

        // 获取任务配置文件并解析 也即 job.xml
        final JobConf job = new JobConf(MRJobConfig.JOB_CONF_FILE);
        // Initing with our JobConf allows us to avoid loading confs twice
        // 初始化配置避免加载两次
        Limits.init(job);

        UserGroupInformation.setConfiguration(job);
        // MAPREDUCE-6565: need to set configuration for SecurityUtil.
        SecurityUtil.setConfiguration(job);

        /**
         * exec /bin/bash -c
         * "$JAVA_HOME/bin/java
         * -Djava.net.preferIPv4Stack=true
         * -Dhadoop.metrics.log.level=WARN
         * -Xmx820m
         * -Djava.io.tmpdir=$PWD/tmp
         * -Dlog4j.configuration=container-log4j.properties
         * -Dyarn.app.container.log.dir=/opt/app/hadoop-
3.1.3/logs/userlogs/application_1684656010852_0007/container_1684656010852_0007_01_0
00002
         * -Dyarn.app.container.log.filesize=0
         * -Dhadoop.root.logger=INFO,CLA
         * -Dhadoop.root.logfile=syslog
         * org.apache.hadoop.mapred.YarnChild
         * 192.168.6.102
         * 39956
         * attempt_1684656010852_0007_m_000000_0
         * 2
         * 1>/opt/app/hadoop-
3.1.3/logs/userlogs/application_1684656010852_0007/container_1684656010852_0007_01_0
00002/stdout
```

```
         * 2>/opt/app/hadoop-
3.1.3/logs/userlogs/application_1684656010852_0007/container_1684656010852_0007_01_0
00002/stderr
         */
        // AM 地址
        String host = args[0];
        int port = Integer.parseInt(args[1]);

        final InetSocketAddress address =
                NetUtils.createSocketAddrForHost(host, port);
        final TaskAttemptID firstTaskid = TaskAttemptID.forName(args[2]);
        long jvmIdLong = Long.parseLong(args[3]);
        JVMId jvmId = new JVMId(firstTaskid.getJobID(),
                firstTaskid.getTaskType() == TaskType.MAP, jvmIdLong);

        CallerContext.setCurrent(
                new CallerContext.Builder("mr_" + firstTaskid.toString()).build());

        // initialize metrics
        DefaultMetricsSystem.initialize(
                StringUtils.camelize(firstTaskid.getTaskType().name()) + "Task");

        // Security framework already loaded the tokens into current ugi
        Credentials credentials =
                UserGroupInformation.getCurrentUser().getCredentials();

        // Executing with tokens:
        // [Kind: mapreduce.job, Service: job_1684656010852_0007,
        // Ident:
(org.apache.hadoop.mapreduce.security.token.JobTokenIdentifier@22555ebf)]
        LOG.info("Executing with tokens: {}", credentials.getAllTokens());

        // Create TaskUmbilicalProtocol as actual task owner.
        UserGroupInformation taskOwner =

 UserGroupInformation.createRemoteUser(firstTaskid.getJobID().toString());
        Token<JobTokenIdentifier> jt = TokenCache.getJobToken(credentials);
        SecurityUtil.setTokenService(jt, address);
        taskOwner.addToken(jt);
        // 获取 AM(MRAppMaster) 的 Task 监听服务 TaskAttemptListenerImpl
        // 该 TaskAttemptListenerImpl 内部启动了一个 RPC Server 通讯协议为
TaskUmbilicalProtocol
        // 这个获取 AM 的 RPC 服务 TaskAttemptListenerImpl 的 RPC 代理对象 (也即客户端)
        final TaskUmbilicalProtocol umbilical =
                taskOwner.doAs(new PrivilegedExceptionAction<TaskUmbilicalProtocol>
() {
                    @Override
                    public TaskUmbilicalProtocol run() throws Exception {
                        return (TaskUmbilicalProtocol)
RPC.getProxy(TaskUmbilicalProtocol.class,
                                TaskUmbilicalProtocol.versionID, address, job);
                    }
```

```
                });

        // report non-pid to application master
        JvmContext context = new JvmContext(jvmId, "-1000");
        LOG.debug("PID: " + System.getenv().get("JVM_PID"));
        Task task = null;
        UserGroupInformation childUGI = null;
        ScheduledExecutorService logSyncer = null;

        try {
            int idleLoopCount = 0;
            JvmTask myTask = null;
            // poll for new task
            for (int idle = 0; null == myTask; ++idle) {
                long sleepTimeMilliSecs = Math.min(idle * 500, 1500);
                // Sleeping for 0ms before retrying again. Got null now.
                LOG.info("Sleeping for " + sleepTimeMilliSecs
                        + "ms before retrying again. Got null now.");
                MILLISECONDS.sleep(sleepTimeMilliSecs);
                // 发送 RPC 请求给 AM 的 TaskAttemptListenerImpl RPC 服务获取当前
YarnChild 进程执行任务
                myTask = umbilical.getTask(context);
            }
            if (myTask.shouldDie()) {
                return;
            }

            // 获取执行任务 可能是 MapTask 也可能是 ReduceTask
            task = myTask.getTask();
            YarnChild.taskid = task.getTaskID();

            // Create the job-conf and set credentials
            // 配置执行 Task 的一些配置信息以及证书相关的
            configureTask(job, task, credentials, jt);

            // log the system properties
            /**
             * /************************************************************
             * [system properties]
             * os.name: Linux
             * os.version: 3.10.0-1062.el7.x86_64
             * java.home: /opt/app/jdk1.8.0_212/jre
             * java.runtime.version: 1.8.0_212-b10
             * java.vendor: Oracle Corporation
             * java.version: 1.8.0_212
             * java.vm.name: Java HotSpot(TM) 64-Bit Server VM
```

```
            * java.class.path: /opt/app/hadoop-3.1.3/data/nm-local-
dir/usercache/tanbs/appcache/application_1684656010852_0007/container_1684656010852_
0007_01_000002:/opt/app/hadoop-3.1.3/etc/hadoop:/opt/app/hadoop-
3.1.3/share/hadoop/common/hadoop-common-3.1.3-tests.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/hadoop-common-3.1.3.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/hadoop-kms-3.1.3.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/hadoop-nfs-3.1.3.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/jetty-io-9.3.24.v20180605.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/accessors-smart-1.2.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/jcip-annotations-1.0-1.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/animal-sniffer-annotations-1.17.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/netty-3.10.5.Final.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/asm-5.0.4.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/jersey-core-1.19.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/audience-annotations-0.5.0.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/nimbus-jose-jwt-4.41.1.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/avro-1.7.7.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/jetty-security-9.3.24.v20180605.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/checker-qual-2.5.2.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/jetty-server-9.3.24.v20180605.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/commons-beanutils-1.9.3.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/paranamer-2.3.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/commons-cli-1.2.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/jsr311-api-1.1.1.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/commons-codec-1.11.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/jersey-json-1.19.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/commons-collections-3.2.2.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/jul-to-slf4j-1.7.25.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/commons-compress-1.18.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/jersey-servlet-1.19.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/commons-configuration2-2.1.1.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/protobuf-java-2.5.0.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/commons-io-2.5.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/re2j-1.1.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/commons-lang-2.6.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/kerb-admin-1.0.1.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/commons-lang3-3.4.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/kerb-client-1.0.1.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/commons-logging-1.1.3.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/kerb-common-1.0.1.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/commons-math3-3.1.1.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/slf4j-api-1.7.25.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/commons-net-3.6.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/kerb-core-1.0.1.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/curator-client-2.13.0.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/kerb-crypto-1.0.1.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/curator-framework-2.13.0.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/kerb-identity-1.0.1.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/curator-recipes-2.13.0.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/jersey-server-1.19.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/error_prone_annotations-2.2.0.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/kerb-server-1.0.1.jar:/opt/app/hadoop-
```

```
3.1.3/share/hadoop/common/lib/failureaccess-1.0.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/slf4j-log4j12-1.7.25.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/gson-2.2.4.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/snappy-java-1.0.5.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/guava-27.0-jre.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/kerb-simplekdc-1.0.1.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/hadoop-annotations-3.1.3.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/kerb-util-1.0.1.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/hadoop-auth-3.1.3.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/jsch-0.1.54.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/htrace-core4-4.1.0-incubating.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/stax2-api-3.1.4.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/httpclient-4.5.2.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/token-provider-1.0.1.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/httpcore-4.4.4.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/kerby-asn1-1.0.1.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/j2objc-annotations-1.1.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/jettison-1.1.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/jackson-annotations-2.7.8.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/kerby-config-1.0.1.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/jackson-core-2.7.8.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/kerby-pkix-1.0.1.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/jackson-core-asl-1.9.13.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/kerby-util-1.0.1.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/jackson-databind-2.7.8.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/kerby-xdr-1.0.1.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/jackson-jaxrs-1.9.13.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/jetty-http-9.3.24.v20180605.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/jackson-mapper-asl-1.9.13.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/jackson-xc-1.9.13.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/log4j-1.2.17.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/javax.servlet-api-3.1.0.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/woodstox-core-5.0.3.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/jaxb-api-2.2.11.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/metrics-core-3.2.4.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/jaxb-impl-2.2.3-1.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/jetty-servlet-9.3.24.v20180605.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/json-smart-2.3.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/jetty-util-9.3.24.v20180605.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/jsp-api-2.1.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/jetty-webapp-9.3.24.v20180605.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/jsr305-3.0.0.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/jetty-xml-9.3.24.v20180605.jar:/opt/app/hadoop-
3.1.3/share/hadoop/common/lib/listenablefuture-9999.0-empty-to-avoid-conflict-with-
guava.jar:/opt/app/hadoop-3.1.3/share/hadoop/common/lib/zookeeper-
3.4.13.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/hadoop-hdfs-3.1.3-
tests.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/hadoop-hdfs-
3.1.3.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/hadoop-hdfs-client-3.1.3-
tests.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/hadoop-hdfs-client-
3.1.3.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/hadoop-hdfs-httpfs-
3.1.3.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/hadoop-hdfs-native-client-3.1.3-
tests.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/hadoop-hdfs-native-client-
```

```
3.1.3.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/hadoop-hdfs-nfs-
3.1.3.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/hadoop-hdfs-rbf-3.1.3-
tests.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/hadoop-hdfs-rbf-
3.1.3.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/jetty-http-
9.3.24.v20180605.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/accessors-smart-
1.2.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/jaxb-impl-2.2.3-
1.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/animal-sniffer-annotations-
1.17.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/log4j-
1.2.17.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/asm-
5.0.4.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/jcip-annotations-1.0-
1.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/audience-annotations-
0.5.0.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/netty-
3.10.5.Final.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/avro-
1.7.7.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/jetty-io-
9.3.24.v20180605.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/checker-qual-
2.5.2.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/jetty-security-
9.3.24.v20180605.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/commons-beanutils-
1.9.3.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/netty-all-
4.0.52.Final.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/commons-cli-
1.2.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/json-smart-
2.3.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/commons-codec-
1.11.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/jersey-core-
1.19.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/commons-collections-
3.2.2.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/jsr305-
3.0.0.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/commons-compress-
1.18.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/jersey-server-
1.19.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/commons-configuration2-
2.1.1.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/jsr311-api-
1.1.1.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/commons-daemon-
1.0.13.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/nimbus-jose-jwt-
4.41.1.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/commons-io-
2.5.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/okhttp-
2.7.5.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/commons-lang-
2.6.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/kerb-admin-
1.0.1.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/commons-lang3-
3.4.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/kerb-client-
1.0.1.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/commons-logging-
1.1.3.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/kerb-common-
1.0.1.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/commons-math3-
3.1.1.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/okio-
1.6.0.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/commons-net-
3.6.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/kerb-core-
1.0.1.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/curator-client-
2.13.0.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/kerb-crypto-
1.0.1.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/curator-framework-
2.13.0.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/kerb-identity-
1.0.1.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/curator-recipes-
2.13.0.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/jersey-json-
1.19.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/error_prone_annotations-
2.2.0.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/kerb-server-
1.0.1.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/failureaccess-
1.0.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/paranamer-
```

```
2.3.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/gson-2.2.4.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/protobuf-java-2.5.0.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/guava-27.0-jre.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/kerb-simplekdc-1.0.1.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/hadoop-annotations-3.1.3.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/kerb-util-1.0.1.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/hadoop-auth-3.1.3.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/jetty-webapp-9.3.24.v20180605.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/htrace-core4-4.1.0-incubating.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/re2j-1.1.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/httpclient-4.5.2.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/snappy-java-1.0.5.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/httpcore-4.4.4.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/kerby-asn1-1.0.1.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/j2objc-annotations-1.1.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/jersey-servlet-1.19.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/jackson-annotations-2.7.8.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/kerby-config-1.0.1.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/jackson-core-2.7.8.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/kerby-pkix-1.0.1.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/jackson-core-asl-1.9.13.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/kerby-util-1.0.1.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/jackson-databind-2.7.8.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/kerby-xdr-1.0.1.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/jackson-jaxrs-1.9.13.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/jettison-1.1.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/jackson-mapper-asl-1.9.13.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/leveldbjni-all-1.8.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/jackson-xc-1.9.13.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/javax.servlet-api-3.1.0.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/stax2-api-3.1.4.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/jaxb-api-2.2.11.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/jetty-server-9.3.24.v20180605.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/jetty-xml-9.3.24.v20180605.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/jetty-servlet-9.3.24.v20180605.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/jsch-0.1.54.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/jetty-util-9.3.24.v20180605.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/json-simple-1.1.1.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/jetty-util-ajax-9.3.24.v20180605.jar:/opt/app/hadoop-
3.1.3/share/hadoop/hdfs/lib/listenablefuture-9999.0-empty-to-avoid-conflict-with-
guava.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/token-provider-
1.0.1.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/woodstox-core-
5.0.3.jar:/opt/app/hadoop-3.1.3/share/hadoop/hdfs/lib/zookeeper-
3.4.13.jar:/opt/app/hadoop-3.1.3/share/hadoop/yarn/hadoop-yarn-api-
3.1.3.jar:/opt/app/hadoop-3.1.3/share/hadoop/yarn/hadoop-yarn-applications-
distributedshell-3.1.3.jar:/opt/app/hadoop-3.1.3/share/hadoop/yarn/hadoop-yarn-
applications-unmanaged-am-launcher-3.1.3.jar:/opt/app/hadoop-
3.1.3/share/hadoop/yarn/hadoop-yarn-client-3.1.3.jar:/opt/app/hadoop-
3.1.3/share/hadoop/yarn/hadoop-yarn-common-3.1.3.jar:/opt/app/hadoop-
3.1.3/share/hadoop/yarn/hadoop-yarn-registry-3.1.3.jar:/opt/app/hadoop-
3.1.3/share/hadoop/yarn/hadoop-yarn-server-applicationhistoryservice-
3.1.3.jar:/opt/app/hadoop-3.1.3/share/hadoop/yarn/hadoop-yarn-server-common-
```

```
3.1.3.jar:/opt/app/hadoop-3.1.3/share/hadoop/yarn/hadoop-yarn-server-nodemanager-
3.1.3.jar:/opt/app/hadoop-3.1.3/share/hadoop/yarn/hadoop-yarn-server-
resourcemanager-3.1.3.jar:/opt/app/hadoop-3.1.3/share/hadoop/yarn/hadoop-yarn-
server-router-3.1.3.jar:/opt/app/hadoop-3.1.3/share/hadoop/yarn/hadoop-yarn-server-
sharedcachemanager-3.1.3.jar:/opt/app/hadoop-3.1.3/share/hadoop/yarn/hadoop-yarn-
server-tests-3.1.3.jar:/opt/app/hadoop-3.1.3/share/hadoop/yarn/hadoop-yarn-server-
timeline-pluginstorage-3.1.3.jar:/opt/app/hadoop-3.1.3/share/hadoop/yarn/hadoop-
yarn-server-web-proxy-3.1.3.jar:/opt/app/hadoop-3.1.3/share/hadoop/yarn/hadoop-yarn-
services-api-3.1.3.jar:/opt/app/hadoop-3.1.3/share/hadoop/yarn/hadoop-yarn-services-
core-3.1.3.jar:/opt/app/hadoop-3.1.3/share/hadoop/yarn/lib/HikariCP-java7-
2.4.12.jar:/opt/app/hadoop-3.1.3/share/hadoop/yarn/lib/aopalliance-
1.0.jar:/opt/app/hadoop-3.1.3/share/hadoop/yarn/lib/dnsjava-
2.1.7.jar:/opt/app/hadoop-3.1.3/share/hadoop/yarn/lib/ehcache-
3.3.1.jar:/opt/app/hadoop-3.1.3/share/hadoop/yarn/lib/fst-2.50.jar:/opt/app/hadoop-
3.1.3/share/hadoop/yarn/lib/geronimo-jcache_1.0_spec-1.0-alpha-
1.jar:/opt/app/hadoop-3.1.3/share/hadoop/yarn/lib/guice-4.0.jar:/opt/app/hadoop-
3.1.3/share/hadoop/yarn/lib/guice-servlet-4.0.jar:/opt/app/hadoop-
3.1.3/share/hadoop/yarn/lib/jackson-jaxrs-base-2.7.8.jar:/opt/app/hadoop-
3.1.3/share/hadoop/yarn/lib/jackson-jaxrs-json-provider-2.7.8.jar:/opt/app/hadoop-
3.1.3/share/hadoop/yarn/lib/jackson-module-jaxb-annotations-
2.7.8.jar:/opt/app/hadoop-3.1.3/share/hadoop/yarn/lib/java-util-
1.9.0.jar:/opt/app/hadoop-3.1.3/share/hadoop/yarn/lib/javax.inject-
1.jar:/opt/app/hadoop-3.1.3/share/hadoop/yarn/lib/jersey-client-
1.19.jar:/opt/app/hadoop-3.1.3/share/hadoop/yarn/lib/jersey-guice-
1.19.jar:/opt/app/hadoop-3.1.3/share/hadoop/yarn/lib/json-io-
2.5.1.jar:/opt/app/hadoop-3.1.3/share/hadoop/yarn/lib/metrics-core-
3.2.4.jar:/opt/app/hadoop-3.1.3/share/hadoop/yarn/lib/mssql-jdbc-
6.2.1.jre7.jar:/opt/app/hadoop-3.1.3/share/hadoop/yarn/lib/objenesis-
1.0.jar:/opt/app/hadoop-3.1.3/share/hadoop/yarn/lib/snakeyaml-
1.16.jar:/opt/app/hadoop-3.1.3/share/hadoop/yarn/lib/swagger-annotations-
1.5.4.jar:/opt/app/hadoop-3.1.3/share/hadoop/mapreduce/hadoop-mapreduce-client-app-
3.1.3.jar:/opt/app/hadoop-3.1.3/share/hadoop/mapreduce/hadoop-mapreduce-client-
common-3.1.3.jar:/opt/app/hadoop-3.1.3/share/hadoop/mapreduce/hadoop-mapreduce-
client-core-3.1.3.jar:/opt/app/hadoop-3.1.3/share/hadoop/mapreduce/hadoop-mapreduce-
client-hs-3.1.3.jar:/opt/app/hadoop-3.1.3/share/hadoop/mapreduce/hadoop-mapreduce-
client-hs-plugins-3.1.3.jar:/opt/app/hadoop-3.1.3/share/hadoop/mapreduce/hadoop-
mapreduce-client-jobclient-3.1.3-tests.jar:/opt/app/hadoop-
3.1.3/share/hadoop/mapreduce/hadoop-mapreduce-client-jobclient-
3.1.3.jar:/opt/app/hadoop-3.1.3/share/hadoop/mapreduce/hadoop-mapreduce-client-
nativetask-3.1.3.jar:/opt/app/hadoop-3.1.3/share/hadoop/mapreduce/hadoop-mapreduce-
client-shuffle-3.1.3.jar:/opt/app/hadoop-3.1.3/share/hadoop/mapreduce/hadoop-
mapreduce-client-uploader-3.1.3.jar:/opt/app/hadoop-
3.1.3/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.1.3.jar:/opt/app/hadoop-
3.1.3/share/hadoop/mapreduce/lib/hamcrest-core-1.3.jar:/opt/app/hadoop-
3.1.3/share/hadoop/mapreduce/lib/junit-
4.11.jar:job.jar/job.jar:job.jar/classes/:job.jar/lib/*:/opt/app/hadoop-
3.1.3/data/nm-local-
dir/usercache/tanbs/appcache/application_1684656010852_0007/container_1684656010852_
0007_01_000002/job.jar
            * java.io.tmpdir: /opt/app/hadoop-3.1.3/data/nm-local-
dir/usercache/tanbs/appcache/application_1684656010852_0007/container_1684656010852_
0007_01_000002/tmp
```

```
             * user.dir: /opt/app/hadoop-3.1.3/data/nm-local-
dir/usercache/tanbs/appcache/application_1684656010852_0007/container_1684656010852_
0007_01_000002
             * user.name: tanbs
             * ************************************************************
             */
            String systemPropsToLog = MRApps.getSystemPropertiesToLog(job);
            if (systemPropsToLog != null) {
                LOG.info(systemPropsToLog);
            }

            // Initiate Java VM metrics
            JvmMetrics.initSingleton(jvmId.toString(), job.getSessionId());
            childUGI = UserGroupInformation.createRemoteUser(System
                    .getenv(ApplicationConstants.Environment.USER.toString()));
            // Add tokens to new user so that it may execute its task correctly.
            childUGI.addCredentials(credentials);

            // set job classloader if configured before invoking the task
            // 默认没有配置类加载器
            MRApps.setJobClassLoader(job);

            logSyncer = TaskLog.createLogSyncer();

            // Create a final reference to the task for the doAs block
            // 这里就是执行 MR 任务的最终 MapTask 或者 ReduceTask
            final Task taskFinal = task;
            childUGI.doAs(new PrivilegedExceptionAction<Object>() {
                @Override
                public Object run() throws Exception {
                    // use job-specified working directory
                    setEncryptedSpillKeyIfRequired(taskFinal);

 FileSystem.get(job).setWorkingDirectory(job.getWorkingDirectory());
                    // 调用 MapTask.run() 或者 ReduceTask.run()
                    taskFinal.run(job, umbilical); // run the task
                    return null;
                }
            });
        } catch (FSError e) {
            LOG.error("FSError from child", e);
            if (!ShutdownHookManager.get().isShutdownInProgress()) {
                umbilical.fsError(taskid, e.getMessage());
            }
        } catch (Exception exception) {
            LOG.warn("Exception running child : "
                    + StringUtils.stringifyException(exception));
            try {
                if (task != null) {
                    // do cleanup for the task
                    if (childUGI == null) { // no need to job into doAs block
                        task.taskCleanup(umbilical);
```

```
                    } else {
                        final Task taskFinal = task;
                        childUGI.doAs(new PrivilegedExceptionAction<Object>() {
                            @Override
                            public Object run() throws Exception {
                                taskFinal.taskCleanup(umbilical);
                                return null;
                            }
                        });
                    }
                }
            } catch (Exception e) {
                LOG.info("Exception cleaning up: " +
StringUtils.stringifyException(e));
            }
            // Report back any failures, for diagnostic purposes
            if (taskid != null) {
                if (!ShutdownHookManager.get().isShutdownInProgress()) {
                    umbilical.fatalError(taskid,
                            StringUtils.stringifyException(exception), false);
                }
            }
        } catch (Throwable throwable) {
            LOG.error("Error running child : "
                    + StringUtils.stringifyException(throwable));
            if (taskid != null) {
                if (!ShutdownHookManager.get().isShutdownInProgress()) {
                    Throwable tCause = throwable.getCause();
                    String cause =
                            tCause == null ? throwable.getMessage() : StringUtils
                                    .stringifyException(tCause);
                    umbilical.fatalError(taskid, cause, false);
                }
            }
        } finally {
            RPC.stopProxy(umbilical);
            DefaultMetricsSystem.shutdown();
            TaskLog.syncLogsShutdown(logSyncer);
        }
    }
}
```

6.1 获取 AM(MRAppMaster) 的 Task 监听服务 TaskAttemptListenerImpl RPC 服务客户端代理

```
// 获取 AM(MRAppMaster) 的 Task 监听服务 TaskAttemptListenerImpl
        // 该 TaskAttemptListenerImpl 内部启动了一个 RPC Server 通讯协议为
TaskUmbilicalProtocol
        // 这个获取 AM 的 RPC 服务 TaskAttemptListenerImpl 的 RPC 代理对象 (也即客户端)
        final TaskUmbilicalProtocol umbilical =
                taskOwner.doAs(new PrivilegedExceptionAction<TaskUmbilicalProtocol>
() {
                    @Override
                    public TaskUmbilicalProtocol run() throws Exception {
                        return (TaskUmbilicalProtocol)
RPC.getProxy(TaskUmbilicalProtocol.class,
                                TaskUmbilicalProtocol.versionID, address, job);
                    }
                });
```

6.2 发送 RPC 请求 TaskAttemptListenerImpl RPC 服务获取执行 Task

```
try {
            int idleLoopCount = 0;
            JvmTask myTask = null;
            // poll for new task
            for (int idle = 0; null == myTask; ++idle) {
                long sleepTimeMilliSecs = Math.min(idle * 500, 1500);
                // Sleeping for 0ms before retrying again. Got null now.
                LOG.info("Sleeping for " + sleepTimeMilliSecs
                        + "ms before retrying again. Got null now.");
                MILLISECONDS.sleep(sleepTimeMilliSecs);
                // 发送 RPC 请求给 AM 的 TaskAttemptListenerImpl RPC 服务获取当前
YarnChild 进程执行任务
                myTask = umbilical.getTask(context);
            }
```

6.3 执行 MapTask或者ReduceTask 业务逻辑

```
// Create a final reference to the task for the doAs block
            // 这里就是执行 MR 任务的最终 MapTask 或者 ReduceTask
            final Task taskFinal = task;
            childUGI.doAs(new PrivilegedExceptionAction<Object>() {
                @Override
                public Object run() throws Exception {
                    // use job-specified working directory
                    setEncryptedSpillKeyIfRequired(taskFinal);

 FileSystem.get(job).setWorkingDirectory(job.getWorkingDirectory());
                    // 调用 MapTask.run() 或者 ReduceTask.run()
                    taskFinal.run(job, umbilical); // run the task
                    return null;
                }
```

```
            });
```

6.3.1 执行 Mapper 调用 MapTask.run()

```
/**
 * A Map task.
 */
@InterfaceAudience.LimitedPrivate({"MapReduce"})
@InterfaceStability.Unstable
public class MapTask extends Task {
 @Override
    public void run(final JobConf job, final TaskUmbilicalProtocol umbilical)
            throws IOException, ClassNotFoundException, InterruptedException {
        this.umbilical = umbilical;

        // MapTask
        if (isMapTask()) {
            // If there are no reducers then there won't be any sort. Hence the map
            // phase will govern the entire attempt's progress.
            if (conf.getNumReduceTasks() == 0) {
                mapPhase = getProgress().addPhase("map", 1.0f);
            } else {
                // If there are reducers then the entire attempt's progress will be
                // split between the map phase (67%) and the sort phase (33%).
                mapPhase = getProgress().addPhase("map", 0.667f);
                sortPhase = getProgress().addPhase("sort", 0.333f);
            }
        }
        TaskReporter reporter = startReporter(umbilical);

        // 默认使用 NEW API
        boolean useNewApi = job.getUseNewMapper();
        // 初始化 MapTask
        initialize(job, getJobID(), reporter, useNewApi);

        // check if it is a cleanupJobTask
        if (jobCleanup) {
            runJobCleanupTask(umbilical, reporter);
            return;
        }
        if (jobSetup) {
            runJobSetupTask(umbilical, reporter);
            return;
        }
        if (taskCleanup) {
            runTaskCleanupTask(umbilical, reporter);
            return;
        }

        if (useNewApi) {
            // 这里就是最底层执行 Mapper 业务逻辑入口
```

```
            runNewMapper(job, splitMetaInfo, umbilical, reporter);
        } else {
            runOldMapper(job, splitMetaInfo, umbilical, reporter);
        }
        done(umbilical, reporter);
    }
}
```

```
@SuppressWarnings("unchecked")
    private <INKEY, INVALUE, OUTKEY, OUTVALUE>
    void runNewMapper(final JobConf job,
                      final TaskSplitIndex splitIndex,
                      final TaskUmbilicalProtocol umbilical,
                      TaskReporter reporter
    ) throws IOException, ClassNotFoundException,
            InterruptedException {
        // make a task context so we can get the classes
        // 创建 Task 上下文 TaskAttemptContextImpl
        org.apache.hadoop.mapreduce.TaskAttemptContext taskContext =
                new org.apache.hadoop.mapreduce.task.TaskAttemptContextImpl(job,
                        getTaskID(),
                        reporter);
        // make a mapper
        // 获取用户自定义 Mapper 类
        org.apache.hadoop.mapreduce.Mapper<INKEY, INVALUE, OUTKEY, OUTVALUE> mapper
=
                (org.apache.hadoop.mapreduce.Mapper<INKEY, INVALUE, OUTKEY,
OUTVALUE>)
                        ReflectionUtils.newInstance(taskContext.getMapperClass(),
job);
        // make the input format
        // 获取用户自定义输入类 默认 TextInputFormat
        org.apache.hadoop.mapreduce.InputFormat<INKEY, INVALUE> inputFormat =
                (org.apache.hadoop.mapreduce.InputFormat<INKEY, INVALUE>)

 ReflectionUtils.newInstance(taskContext.getInputFormatClass(), job);
        // rebuild the input split
        // 获取任务切片信息并解析当前 MapTask 处理哪些数据
        org.apache.hadoop.mapreduce.InputSplit split = null;
        split = getSplitDetails(new Path(splitIndex.getSplitLocation()),
                splitIndex.getStartOffset());
        // hdfs://hadoop102:8020/wordcount/input/LICENSE.txt:0+147145
        LOG.info("Processing split: " + split);

        // 根据输入类型以及切片信息构建创建读取数据对象 NewTrackingRecordReader
        org.apache.hadoop.mapreduce.RecordReader<INKEY, INVALUE> input =
                new NewTrackingRecordReader<INKEY, INVALUE>
                        (split, inputFormat, reporter, taskContext);

        job.setBoolean(JobContext.SKIP_RECORDS, isSkipping());
        org.apache.hadoop.mapreduce.RecordWriter output = null;
```

```
        // get an output object
        // 判断 ReduceTask 个数的情况
        if (job.getNumReduceTasks() == 0) {
            output =
                    new NewDirectOutputCollector(taskContext, job, umbilical,
reporter);
        } else {
            output = new NewOutputCollector(taskContext, job, umbilical, reporter);
        }

        // 创建 MapTask 上下文对象 MapContextImpl
        org.apache.hadoop.mapreduce.MapContext<INKEY, INVALUE, OUTKEY, OUTVALUE>
                mapContext =
                new MapContextImpl<INKEY, INVALUE, OUTKEY, OUTVALUE>(job,
getTaskID(),
                        input, output,
                        committer,
                        reporter, split);

        // 包装 MapTask 上下文对象 MapContextImpl
        org.apache.hadoop.mapreduce.Mapper<INKEY, INVALUE, OUTKEY, OUTVALUE>.Context
                mapperContext =
                new WrappedMapper<INKEY, INVALUE, OUTKEY, OUTVALUE>().getMapContext(
                        mapContext);

        try {
            // 初始化输入数据环境 调用 NewTrackingRecordReader.initialize()
            input.initialize(split, mapperContext);
            // 调用 Mapper.run() 执行读取数据处理业务逻辑
            mapper.run(mapperContext);
            mapPhase.complete();
            setPhase(TaskStatus.Phase.SORT);
            statusUpdate(umbilical);
            input.close();
            input = null;
            output.close(mapperContext);
            output = null;
        } finally {
            closeQuietly(input);
            closeQuietly(output, mapperContext);
        }
    }
```

6.3.1 执行 Reducer调用 ReduceTask.run()

```
/**
 * A Reduce task.
 */
@InterfaceAudience.Private
@InterfaceStability.Unstable
```

```
public class ReduceTask extends Task {
  @Override
    @SuppressWarnings("unchecked")
    public void run(JobConf job, final TaskUmbilicalProtocol umbilical)
            throws IOException, InterruptedException, ClassNotFoundException {
        job.setBoolean(JobContext.SKIP_RECORDS, isSkipping());

        if (isMapOrReduce()) {
            copyPhase = getProgress().addPhase("copy");
            sortPhase = getProgress().addPhase("sort");
            reducePhase = getProgress().addPhase("reduce");
        }
        // start thread that will handle communication with parent
        TaskReporter reporter = startReporter(umbilical);

        // 默认 true
        boolean useNewApi = job.getUseNewReducer();
        initialize(job, getJobID(), reporter, useNewApi);

        // check if it is a cleanupJobTask
        if (jobCleanup) {
            runJobCleanupTask(umbilical, reporter);
            return;
        }
        if (jobSetup) {
            runJobSetupTask(umbilical, reporter);
            return;
        }
        if (taskCleanup) {
            runTaskCleanupTask(umbilical, reporter);
            return;
        }

        // Initialize the codec
        // 初始化编码器 默认 DefaultCodec
        codec = initCodec();
        RawKeyValueIterator rIter = null;
        ShuffleConsumerPlugin shuffleConsumerPlugin = null;

        // 获取 Combiner 类
        Class combinerClass = conf.getCombinerClass();
        CombineOutputCollector combineCollector =
                (null != combinerClass) ?
                        new CombineOutputCollector(reduceCombineOutputCounter,
reporter, conf) : null;

        Class<? extends ShuffleConsumerPlugin> clazz =
                job.getClass(MRConfig.SHUFFLE_CONSUMER_PLUGIN, Shuffle.class,
ShuffleConsumerPlugin.class);

        shuffleConsumerPlugin = ReflectionUtils.newInstance(clazz, job);
```

```
        // Using ShuffleConsumerPlugin:
org.apache.hadoop.mapreduce.task.reduce.Shuffle@6622fc65
        LOG.info("Using ShuffleConsumerPlugin: " + shuffleConsumerPlugin);

        ShuffleConsumerPlugin.Context shuffleContext =
                new ShuffleConsumerPlugin.Context(getTaskID(), job,
FileSystem.getLocal(job), umbilical,
                        super.lDirAlloc, reporter, codec,
                        combinerClass, combineCollector,
                        spilledRecordsCounter, reduceCombineInputCounter,
                        shuffledMapsCounter,
                        reduceShuffleBytes, failedShuffleCounter,
                        mergedMapOutputsCounter,
                        taskStatus, copyPhase, sortPhase, this,
                        mapOutputFile, localMapFiles);
        shuffleConsumerPlugin.init(shuffleContext);

        rIter = shuffleConsumerPlugin.run();

        // free up the data structures
        mapOutputFilesOnDisk.clear();

        sortPhase.complete();                         // sort is complete
        setPhase(TaskStatus.Phase.REDUCE);
        statusUpdate(umbilical);
        Class keyClass = job.getMapOutputKeyClass();
        Class valueClass = job.getMapOutputValueClass();
        RawComparator comparator = job.getOutputValueGroupingComparator();

        if (useNewApi) {
            // Reducer 执行入口
            runNewReducer(job, umbilical, reporter, rIter, comparator,
                    keyClass, valueClass);
        } else {
            runOldReducer(job, umbilical, reporter, rIter, comparator,
                    keyClass, valueClass);
        }

        shuffleConsumerPlugin.close();
        done(umbilical, reporter);
    }
}
```

```
 @SuppressWarnings("unchecked")
    private <INKEY, INVALUE, OUTKEY, OUTVALUE>
    void runNewReducer(JobConf job,
                       final TaskUmbilicalProtocol umbilical,
                       final TaskReporter reporter,
                       RawKeyValueIterator rIter,
                       RawComparator<INKEY> comparator,
                       Class<INKEY> keyClass,
```

```
                        Class<INVALUE> valueClass
    ) throws IOException, InterruptedException,
            ClassNotFoundException {
        // wrap value iterator to report progress.
        final RawKeyValueIterator rawIter = rIter;
        rIter = new RawKeyValueIterator() {
            public void close() throws IOException {
                rawIter.close();
            }

            public DataInputBuffer getKey() throws IOException {
                return rawIter.getKey();
            }

            public Progress getProgress() {
                return rawIter.getProgress();
            }

            public DataInputBuffer getValue() throws IOException {
                return rawIter.getValue();
            }

            public boolean next() throws IOException {
                boolean ret = rawIter.next();
                reporter.setProgress(rawIter.getProgress().getProgress());
                return ret;
            }
        };
        // make a task context so we can get the classes
        org.apache.hadoop.mapreduce.TaskAttemptContext taskContext =
                new org.apache.hadoop.mapreduce.task.TaskAttemptContextImpl(job,
                        getTaskID(), reporter);
        // make a reducer
        org.apache.hadoop.mapreduce.Reducer<INKEY, INVALUE, OUTKEY, OUTVALUE>
reducer =
                (org.apache.hadoop.mapreduce.Reducer<INKEY, INVALUE, OUTKEY,
OUTVALUE>)
                        ReflectionUtils.newInstance(taskContext.getReducerClass(),
job);
        org.apache.hadoop.mapreduce.RecordWriter<OUTKEY, OUTVALUE> trackedRW =
                new NewTrackingRecordWriter<OUTKEY, OUTVALUE>(this, taskContext);
        job.setBoolean("mapred.skip.on", isSkipping());
        job.setBoolean(JobContext.SKIP_RECORDS, isSkipping());
        org.apache.hadoop.mapreduce.Reducer.Context
                reducerContext = createReduceContext(reducer, job, getTaskID(),
                rIter, reduceInputKeyCounter,
                reduceInputValueCounter,
                trackedRW,
                committer,
                reporter, comparator, keyClass,
                valueClass);
        try {
```

```
            // 执行
            reducer.run(reducerContext);
        } finally {
            trackedRW.close(reducerContext);
        }
    }
```

```
/**
   * Advanced application writers can use the
   * {@link #run(org.apache.hadoop.mapreduce.Reducer.Context)} method to
   * control how the reduce task works.
   */
  public void run(Context context) throws IOException, InterruptedException {
    setup(context);
    try {
      while (context.nextKey()) {
        reduce(context.getCurrentKey(), context.getValues(), context);
        // If a back up store is used, reset it
        Iterator<VALUEIN> iter = context.getValues().iterator();
        if(iter instanceof ReduceContext.ValueIterator) {
          ((ReduceContext.ValueIterator<VALUEIN>)iter).resetBackupStore();
        }
      }
    } finally {
      cleanup(context);
    }
  }
```