```
[root@linux1 spark-standalone]# sbin/start-all.sh
starting org.apache.spark.deploy.master.Master, logging to /opt/module/spark-standalone/logs/spark-root-org.apache.spark.
deploy.master.Master-1-linux1.out
linux1: starting org.apache.spark.deploy.worker.Worker, logging to /opt/module/spark-standalone/logs/spark-root-org.apach
e.spark.deploy.worker.Worker-1-linux1.out
linux3: starting org.apache.spark.deploy.worker.Worker, logging to /opt/module/spark-standalone/logs/spark-root-org.apach
e.spark.deploy.worker.Worker-1-linux3.out
linux2: starting org.apache.spark.deploy.worker.Worker, logging to /opt/module/spark-standalone/logs/spark-root-org.apach
e.spark.deploy.worker.Worker-1-linux2.out
[root@linux1 spark-standalone]#
```

2) 查看三台服务器运行进程

```
================linux1================
3330 Jps
3238 Worker
3163 Master
================linux2================
2966 Jps
2908 Worker
================linux3================
2978 Worker
3036 Jps
```

3) 查看 Master 资源监控 Web UI 界面: http://linux1:8080

3.2.4 提交应用

```
bin/spark-submit \
--class org.apache.spark.examples.SparkPi \
--master spark://linux1:7077 \
./examples/jars/spark-examples_2.12-3.0.0.jar \
10
```

1) --class 表示要执行程序的主类

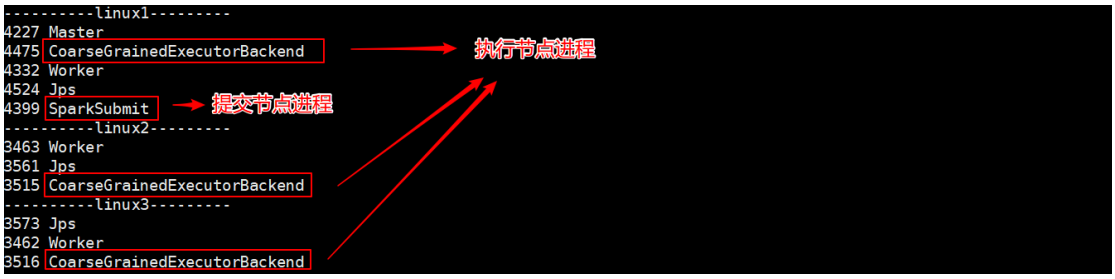
2) --master spark://linux1:7077 独立部署模式，连接到 Spark 集群

3) spark-examples_2.12-3.0.0.jar 运行类所在的 jar 包

4) 数字 10 表示程序的入口参数，用于设定当前应用的任务数量

```
20/06/19 16:42:59 INFO CoarseGrainedSchedulerBackend$DriverEndpoint: Registered executor NettyRpcEndpointRef(spark-client://Executor)
 (192.168.1.101:54494) with ID 0
20/06/19 16:42:59 INFO TaskSetManager: Finished task 3.0 in stage 0.0 (TID 3) in 13420 ms on 192.168.1.102 (executor 1) (9/10)
20/06/19 16:42:59 INFO TaskSetManager: Finished task 2.0 in stage 0.0 (TID 2) in 13431 ms on 192.168.1.102 (executor 1) (10/10)
20/06/19 16:42:59 INFO TaskSchedulerImpl: Removed TaskSet 0.0, whose tasks have all completed, from pool
20/06/19 16:42:59 INFO DAGScheduler: ResultStage 0 (reduce at SparkPi.scala:38) finished in 17.999 s
20/06/19 16:42:59 INFO DAGScheduler: Job 0 is finished. Cancelling potential speculative or zombie tasks for this job
20/06/19 16:42:59 INFO TaskSchedulerImpl: Killing all running tasks in stage 0: Stage finished
20/06/19 16:42:59 INFO DAGScheduler: Job 0 finished: reduce at SparkPi.scala:38, took 18.633508 s
Pi is roughly 3.141943141943142
20/06/19 16:42:59 INFO SparkUI: Stopped Spark web UI at http://linux1:4040
20/06/19 16:42:59 INFO StandaloneSchedulerBackend: Shutting down all executors
20/06/19 16:42:59 INFO CoarseGrainedSchedulerBackend$DriverEndpoint: Asking each executor to shut down
20/06/19 16:42:59 INFO MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
20/06/19 16:43:00 INFO MemoryStore: MemoryStore cleared
20/06/19 16:43:00 INFO BlockManager: BlockManager stopped
20/06/19 16:43:00 INFO BlockManagerMaster: BlockManagerMaster stopped
20/06/19 16:43:00 INFO OutputCommitCoordinator$OutputCommitCoordinatorEndpoint: OutputCommitCoordinator stopped!
20/06/19 16:43:00 INFO SparkContext: Successfully stopped SparkContext
20/06/19 16:43:00 INFO ShutdownHookManager: Shutdown hook called
20/06/19 16:43:00 INFO ShutdownHookManager: Deleting directory /tmp/spark-62035251-f7d8-46d7-95c7-a03c9a08edce
20/06/19 16:43:00 INFO ShutdownHookManager: Deleting directory /tmp/spark-2d166a60-b447-43fb-8115-811f379fea8e
```

执行任务时，会产生多个 Java 进程



执行任务时，默认采用服务器集群节点的总核数，每个节点内存 1024M。

Completed Applications (2)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
app-20200619164410-0001	Spark Pi	8	1024.0 MiB		2020/06/19 16:44:10	root	FINISHED	14 s
app-20200619164212-0000	Spark Pi	8	1024.0 MiB		2020/06/19 16:42:12	root	FINISHED	47 s

3.2.5 提交参数说明

在提交应用中，一般会同时一些提交参数

```
bin/spark-submit \
--class <main-class>
--master <master-url> \
... # other options
<application-jar> \
[application-arguments]
```

参数	解释	可选值举例
--class	Spark 程序中包含主函数的类	
--master	Spark 程序运行的模式(环境)	模式：local[*]、spark://linux1:7077、Yarn
--executor-memory 1G	指定每个 executor 可用内存为 1G	符合集群内存配置即可，具体情况具体分析。
--total-executor-cores 2	指定所有 executor 使用的 cpu 核数为 2 个	
--executor-cores	指定每个 executor 使用的 cpu 核数	
application-jar	打包好的应用 jar，包含依赖。这个 URL 在集群中全局可见。 比如 hdfs:// 共享存储系统，如果是	

	file:// path，那么所有的节点的 path 都包含同样的 jar	
application-arguments	传给 main()方法的参数	

3.2.6 配置历史服务

由于 spark-shell 停止掉后，集群监控 linux1:4040 页面就看不到历史任务的运行情况，所以开发时都配置历史服务器记录任务运行情况。

1) 修改 spark-defaults.conf.template 文件名为 spark-defaults.conf

```
mv spark-defaults.conf.template spark-defaults.conf
```

2) 修改 spark-default.conf 文件，配置日志存储路径

```
spark.eventLog.enabled          true
spark.eventLog.dir              hdfs://linux1:8020/directory
```

注意：需要启动 hadoop 集群，HDFS 上的 directory 目录需要提前存在。

```
sbin/start-dfs.sh
hadoop fs -mkdir /directory
```

3) 修改 spark-env.sh 文件，添加日志配置

```
export SPARK_HISTORY_OPTS="
-Dspark.history.ui.port=18080
-Dspark.history.fs.logDirectory=hdfs://linux1:8020/directory
-Dspark.history.retainedApplications=30"
```

- 参数 1 含义：WEB UI 访问的端口号为 18080
- 参数 2 含义：指定历史服务器日志存储路径
- 参数 3 含义：指定保存 Application 历史记录的个数，如果超过这个值，旧的应用程序信息将被删除，这个是内存中的应用数，而不是页面上显示的应用数。

4) 分发配置文件

```
xsync conf
```

5) 重新启动集群和历史服务

```
sbin/start-all.sh
sbin/start-history-server.sh
```

6) 重新执行任务

```
bin/spark-submit \
--class org.apache.spark.examples.SparkPi \
--master spark://linux1:7077 \
./examples/jars/spark-examples_2.12-3.0.0.jar \
10
```

```
20/06/19 17:41:32 INFO StandaloneSchedulerBackend: Connected to Spark cluster with app ID app-20200619174132-0000
20/06/19 17:41:32 INFO Utils: Successfully started service 'org.apache.spark.network.netty.NettyBlockTransferService' on port 34345.
20/06/19 17:41:32 INFO NettyBlockTransferService: Server created on linux1:34345
20/06/19 17:41:32 INFO BlockManager: Using org.apache.spark.storage.RandomBlockReplicationPolicy for block replication policy
20/06/19 17:41:32 INFO StandaloneAppClient$ClientEndpoint: Executor added: app-20200619174132-0000/0 on worker-20200619174053-192.168
.1.101-33517 (192.168.1.101:33517) with 4 core(s)
20/06/19 17:41:32 INFO StandaloneSchedulerBackend: Granted executor ID app-20200619174132-0000/0 on hostPort 192.168.1.101:33517 with
 4 core(s), 1024.0 MiB RAM
20/06/19 17:41:32 INFO StandaloneAppClient$ClientEndpoint: Executor added: app-20200619174132-0000/1 on worker-20200619174040-192.168
.1.102-44849 (192.168.1.102:44849) with 2 core(s)
20/06/19 17:41:32 INFO StandaloneSchedulerBackend: Granted executor ID app-20200619174132-0000/1 on hostPort 192.168.1.102:44849 with
 2 core(s), 1024.0 MiB RAM
20/06/19 17:41:32 INFO StandaloneAppClient$ClientEndpoint: Executor added: app-20200619174132-0000/2 on worker-20200619174047-192.168
.1.103-33749 (192.168.1.103:33749) with 2 core(s)
20/06/19 17:41:32 INFO StandaloneSchedulerBackend: Granted executor ID app-20200619174132-0000/2 on hostPort 192.168.1.103:33749 with
 2 core(s), 1024.0 MiB RAM
20/06/19 17:41:32 INFO BlockManagerMaster: Registering BlockManager BlockManagerId(driver, linux1, 34345, None)
20/06/19 17:41:32 INFO BlockManagerMasterEndpoint: Registering block manager linux1:34345 with 366.3 MiB RAM, BlockManagerId(driver,
linux1, 34345, None)
20/06/19 17:41:32 INFO BlockManagerMaster: Registered BlockManager BlockManagerId(driver, linux1, 34345, None)
20/06/19 17:41:32 INFO BlockManager: Initialized BlockManager: BlockManagerId(driver, linux1, 34345, None)
```

7) 查看历史服务：http://linux1:18080

3.2.7 配置高可用（HA）

所谓的高可用是因为当前集群中的 Master 节点只有一个，所以会存在单点故障问题。所以为了解决单点故障问题，需要在集群中配置多个 Master 节点，一旦处于活动状态的 Master 发生故障时，由备用 Master 提供服务，保证作业可以继续执行。这里的高可用一般采用 Zookeeper 设置

集群规划:

	Linux1	Linux2	Linux3
Spark	Master Zookeeper Worker	Master Zookeeper Worker	Zookeeper Worker

1) 停止集群

```
sbin/stop-all.sh
```

2) 启动 Zookeeper

```
xstart zk
```

3) 修改 spark-env.sh 文件添加如下配置

```
注释如下内容：
#SPARK_MASTER_HOST=linux1
#SPARK_MASTER_PORT=7077

添加如下内容：
#Master 监控页面默认访问端口为 8080，但是可能会和 Zookeeper 冲突，所以改成 8989，也可以自定义，访问 UI 监控页面时请注意
SPARK_MASTER_WEBUI_PORT=8989

export SPARK_DAEMON_JAVA_OPTS="
```

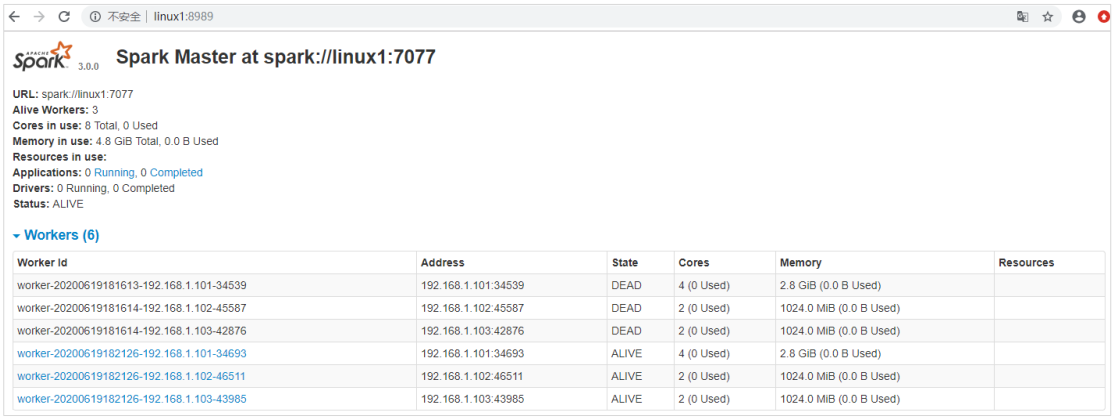
```
-Dspark.deploy.recoveryMode=ZOOKEEPER
-Dspark.deploy.zookeeper.url=linux1,linux2,linux3
-Dspark.deploy.zookeeper.dir=/spark"
```

4) 分发配置文件

```
xsync conf/
```

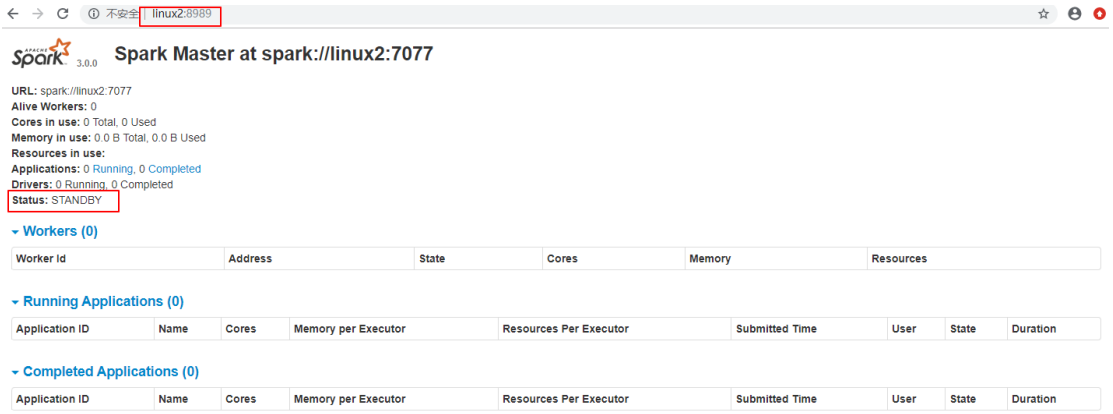
5) 启动集群

```
sbin/start-all.sh
```



6) 启动 linux2 的单独 Master 节点，此时 linux2 节点 Master 状态处于备用状态

```
[root@linux2 spark-standalone]# sbin/start-master.sh
```



7) 提交应用到高可用集群

```
bin/spark-submit \
--class org.apache.spark.examples.SparkPi \
--master spark://linux1:7077,linux2:7077 \
./examples/jars/spark-examples_2.12-3.0.0.jar \
10
```

8) 停止 linux1 的 Master 资源监控进程

```
[root@linux1 spark-standalone]# jps
4673 JobHistoryServer
6802 Worker
6900 Jps
4342 DataNode
4966 QuorumPeerMain
4151 NameNode
4794 NodeManager
6703 Master
[root@linux1 spark-standalone]# kill -9 6703
[root@linux1 spark-standalone]#
```

9) 查看 linux2 的 Master 资源监控 Web UI，稍等一段时间后，linux2 节点的 Master 状态提升为活动状态

3.3 Yarn 模式

独立部署（Standalone）模式由 Spark 自身提供计算资源，无需其他框架提供资源。这种方式降低了和其他第三方资源框架的耦合性，独立性非常强。但是你也要记住，Spark 主要是计算框架，而不是资源调度框架，所以本身提供的资源调度并不是它的强项，所以还是和其他专业的资源调度框架集成会更靠谱一些。所以接下来我们来学习在强大的 Yarn 环境下 Spark 是如何工作的（其实是因为在国内工作中，Yarn 使用的非常多）。

3.3.1 解压缩文件

将 spark-3.0.0-bin-hadoop3.2.tgz 文件上传到 linux 并解压缩，放置在指定位置。

```
tar -zxvf spark-3.0.0-bin-hadoop3.2.tgz -C /opt/module
cd /opt/module
mv spark-3.0.0-bin-hadoop3.2 spark-yarn
```

3.3.2 修改配置文件

1) 修改 hadoop 配置文件/opt/module/hadoop/etc/hadoop/yarn-site.xml，并分发

```
<!--是否启动一个线程检查每个任务正使用的物理内存量，如果任务超出分配值，则直接将其杀掉，默认是 true -->
<property>
    <name>yarn.nodemanager.pmem-check-enabled</name>
    <value>false</value>
</property>

<!--是否启动一个线程检查每个任务正使用的虚拟内存量，如果任务超出分配值，则直接将其杀掉，默认是 true -->
<property>
    <name>yarn.nodemanager.vmem-check-enabled</name>
    <value>false</value>
</property>
```