# Spark 基础

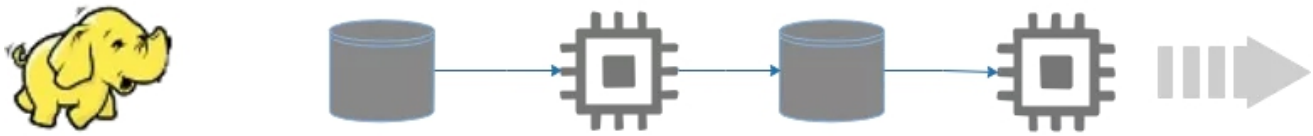
## 一、Spark的介绍

### 1.定义

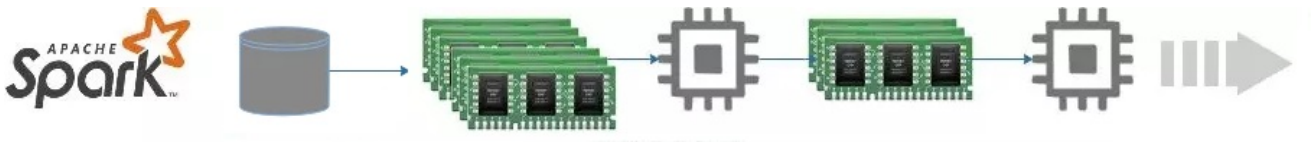
> Spark是一种基于内存的快速、通用、可扩展的大数据分析引擎。

### 2.Spark VS MapReduce

Spark的诞生仅仅是为了替换早期的Hadoop的MapReduce计算引擎。Spark并没有存储解决方案，在Spark的架构中，底层存储方案依然延续Hadooop的HDFS/Hbase.由于Hadoop的MapReduce是大数据时代出现的第一类大数据分析工具，因为出现比较早仅仅为了满足大数据计算的刚性需求（能够做到对大数据的计算，并且可以保证在一个合理的时间范围内）。因此伴随着科技的进步，互联网的快速发展，人们开始对大数据计算提出了更苛刻要求



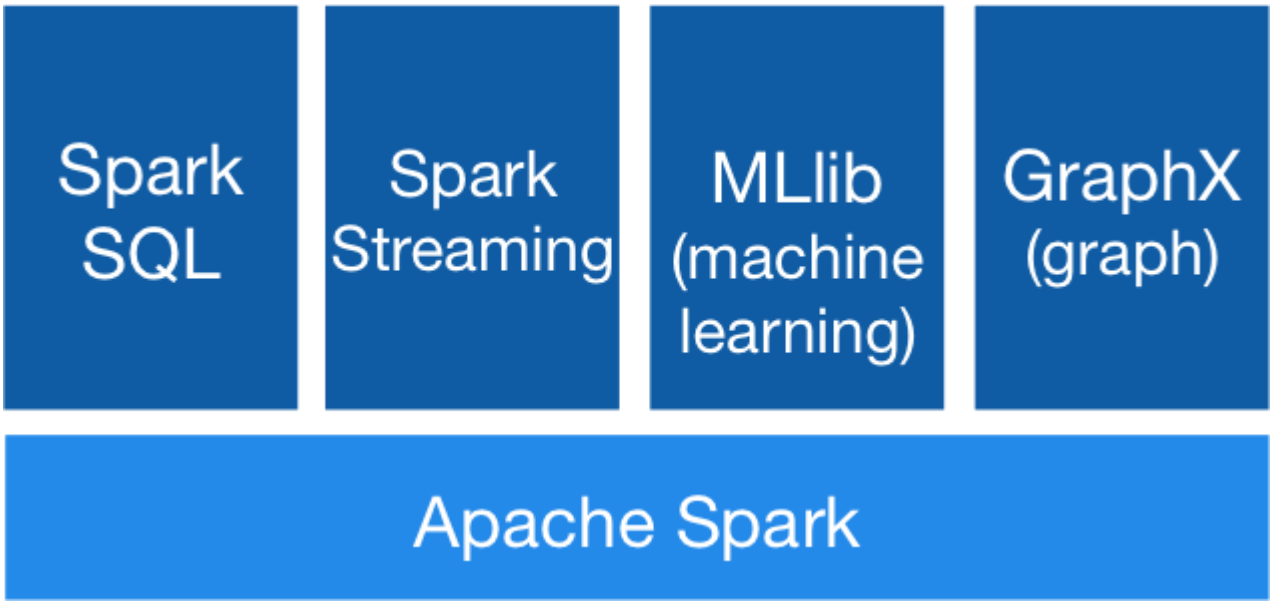
Spark的设计中汲取了Map Reduce的设计经验，在2009 年Spark在加州伯克利AMP实验室诞生，2010年首次开源，2013年6开始在Apache孵化，2014年2月份正式成为Apache顶级项目。由于Spark计算底层引擎使用`批处理`计算模型实现，非常容易被广大深受MapReduce计算折磨程序员所接受，所以就导致了Spark 的关注度直线提升




```
mapreduce编程模型= map + shuffle + reduce
需求：通过单个map+reduce解决不了
解决方案：mr1.jar(map+reduce) --> HDFS ---> mr2.jar(map+reduce)    对多个mr进行编排

.算子().算子().算子()
中间算子的执行结果可以不需要落盘，直接交给后续算子操作，提高开发效率，执行效率
spark执行过程中也有shuffle，也会产生落盘
```

### 3.Spark内置模块




Spark Core：实现了Spark的基本功能，包含任务调度、内存管理、错误恢复、与存储系统交互等模块。Spark Core中还包含了对弹性分布式数据集(Resilient Distributed DataSet，简称RDD)的API定义。

Spark SQL：是Spark用来操作结构化数据的程序包。通过Spark SQL，我们可以使用 SQL或者Apache Hive版本的SQL方言(HQL)来查询数据。Spark SQL支持多种数据源，比如Hive表、Parquet以及JSON等。

Spark Streaming：是Spark提供的对实时数据进行流式计算的组件。提供了用来操作数据流的API，并且与Spark Core中的 RDD API高度对应。

Spark MLlib：提供常见的机器学习(ML)功能的程序库。包括分类、回归、聚类、协同过滤等，还提供了模型评估、数据 导入等额外的支持功能。

集群管理器：Spark 设计为可以高效地在一个计算节点到数千个计算节点之间伸缩计 算。为了实现这样的要求，同时获得最大灵活性，Spark支持在各种集群管理器(Cluster Manager)上运行，包括Hadoop YARN、Apache Mesos，以及Spark自带的一个简易调度 器，叫作独立调度器。

# 二、安装Spark

## 1.Spark相关地址

1．官网地址

http://spark.apache.org/

2．文档查看地址

http://spark.apache.org/docs/2.4.3/

3．下载地址

https://spark.apache.org/downloads.html

# 2.重要角色

## 2.1 Driver（驱动器）

Spark的驱动器是执行开发程序中的main方法的进程。它负责开发人员编写的用来创建SparkContext、创建RDD，以及进行RDD的转化操作和行动操作代码的执行。如果你是用spark shell，那么当你启动Spark shell的时候，系统后台自启了一个Spark驱动器程序，就是在Spark shell中预加载的一个叫作 sc的SparkContext对象。如果驱动器程序终止，那么Spark应用也就结束了。

主要负责：

1）把用户程序转为job

2）跟踪Executor的运行状况

3）为执行器(Executor)节点调度任务(task)

4）UI展示应用运行状况

## 2.2 Executor（执行器）

Spark Executor是一个工作进程，负责在 Spark 作业中运行任务，任务间相互独立。Spark 应用启动时，Executor节点被同时启动，并且始终伴随着整个 Spark 应用的生命周期而存在。如果有Executor节点发生了故障或崩溃，Spark 应用也可以继续执行，会将出错节点上的任务调度到其他Executor节点上继续运行。主要负责：

1）负责运行组成 Spark 应用的任务，并将结果返回给驱动器进程；

2）通过自身的块管理器（Block Manager）为用户程序中要求缓存的RDD提供内存式存储。RDD是直接缓存在Executor进程内的，因此任务可以在运行时充分利用缓存数据加速运算。

https://www.cnblogs.com/itboys/p/7782826.html

# 3. Standalone模式

准备工作：正常安装JDK、Hadoop(启动hdfs)

1）上传并解压spark安装包

```
[root@spark1 modules]# tar -zxf spark-2.4.3-bin-hadoop2.7.tgz  -C /opt/installs
[root@spark1 installs]# mv spark-2.4.3-bin-hadoop2.7 spark2.4.3
```

> spark的目录结构
>
> [root@hadoop10 spark2.4.3]# tree -L 1 . ├── bin #脚本文件目录 ├── conf #配置 ├── data ├── examples ├── jars ├── kubernetes ├── LICENSE ├── licenses ├── NOTICE ├── python ├── R ├── README.md ├── RELEASE ├── sbin └── yarn

2) 修改配置文件

```
[root@spark1 installs]# cd spark2.4.3/conf
[root@spark1 conf]# mv slaves.template slaves
[root@spark1 conf]# mv spark-env.sh.template spark-env.sh
[root@spark1 conf]# vi slaves
#配置Spark集群节点主机名
spark1
[root@spark1 conf]# vi spark-env.sh
#声明Spark集群中Master的主机名和端口号
SPARK_MASTER_HOST=spark1
SPARK_MASTER_PORT=7077
```

3) 在spark中配置JAVA_HOME

```
[root@spark1 conf]# cd ..
[root@spark1 spark]# cd sbin
[root@spark1 sbin]# vi spark-config.sh
#在最后增加 JAVA_HOME 配置
export JAVA_HOME=/opt/installs/jdk1.8
```

4) 启动spark

```
[root@spark1 spark]# sbin/start-all.sh
[root@spark1 spark]# jps              分别查看三个节点的启动结果
2054 Jps
2008 Worker
1933 Master
```

> spark两种测试方式: bin/spark-shell (基于命令行的方式)
>
> bin/spark-submit (基于jar包提交的方式)

6) 方式1

```
新建一个a.txt如下，并且上传到hdfs之上
hello hello hello
world world hello

[root@spark1 spark]# bin/spark-shell --master spark://spark1:7077
scala> sc.textFile("hdfs://spark1:9000/a.txt")
         .flatMap(_.split(" "))
         .map((_,1))
         .groupBy(_._1)
         .map(t=>(t._1,t._2.size))
         .collect
res0: Array[(String, Int)] = Array((hello,4), (world,2))
```

7) 方式2

参考：开发部署第一个Spark程序部分

## 4. JobHistoryServer配置

先拍摄一个快照

1) 修改spark-default.conf.template名称, 修改spark-default.conf文件，开启Log

```
[root@spark1 conf]# mv spark-defaults.conf.template spark-defaults.conf
[root@spark1 conf]# vi spark-defaults.conf
spark.eventLog.enabled           true
spark.eventLog.dir               hdfs://spark1:9000/spark-logs
```

> 注意：HDFS上的目录需要提前存在。
>
> [root@spark1 hadoop-2.9.2]# bin/hdfs dfs -mkdir /spark-logs

2) 修改spark-env.sh文件，添加如下配置

```
[root@spark1 conf]# vi spark-env.sh
SPARK_HISTORY_OPTS="-Dspark.history.fs.logDirectory=hdfs://hadoop10:9000/spark-log"
```

3) 启动对应的服务

```
[root@spark1 spark]# sbin/start-history-server.sh
[root@spark1 spark]# jps
9645 HistoryServer    # 对应启动的进程名称
```

5) 查看历史服务

http://spark1:18080

## 5. Spark On Yarn模式

> yarn这个资源调度器不但可以为hadoop的mapreduce申请资源，同时也可以为spark申请资源，运行spark作业
>
> standalone资源调度器只能运行spark作业

准备工作：正常安装JDK、Hadoop(启动hdfs和yarn)

1) 修改yarn-site.xml文件

```
<!--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>
```

3) 修改spark-env.sh，添加如下配置：

```
注意和standalone的区别，不需要在配置SPARK_MASTER_HOST和SPARK_MASTER_PORT
SPARK_HISTORY_OPTS可以继续使用
[root@spark1 conf]# vi spark-env.sh
#不需要在配置SPARK_MASTER_HOST和SPARK_MASTER_PORT 因为不需要启动master进程
#因为spark的作业要交给hadoop中的yarn资源度器
YARN_CONF_DIR=/opt/installs/hadoop2.9.2/etc/hadoop
```

启动yarn

4) 执行一个程序

```
#方式1
[root@spark1 spark]# bin/spark-shell --master yarn
scala> sc.textFile("hdfs://spark1:9000/a.txt")
          .flatMap(_.split(" "))
          .map((_,1))
          .groupBy(_._1)
          .map(t=>(t._1,t._2.size))
res0: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[6] at map at <console>:25

scala> res0.collect
res1: Array[(String, Int)] = Array((hello,4), (world,2))

#方式2
[root@spark1 spark]# bin/spark-submit --master yarn --class com.baizhi.test1.SparkWordCount
/opt/spark-day1-1.0-SNAPSHOT.jar

jar包对应的程序中设置master为yarn new SparkConf().setMaster("yarn").setAppName("wc")
```

注意：在执行任务前先启动hdfs和yarn

# 三、开发部署第一个Spark程序

Spark Shell仅在测试和验证我们的程序时使用的较多，在生产环境中，通常会在IDE中编写程序，然后打成jar包，然后提交到集群，最常用的是创建一个Maven项目，利用Maven来管理jar包的依赖。

1) 创建一个maven项目，并且导入相关依赖

```
<!-- spark依赖-->
<dependency>
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-core_2.11</artifactId>
    <version>2.4.3</version>
</dependency>
<!-- maven项目对scala编译打包 -->
<plugin>
    <groupId>net.alchim31.maven</groupId>
    <artifactId>scala-maven-plugin</artifactId>
    <version>4.0.1</version>
    <executions>
        <execution>
            <id>scala-compile-first</id>
            <phase>process-resources</phase>
            <goals>
                <goal>add-source</goal>
                <goal>compile</goal>
            </goals>
        </execution>
    </executions>
</plugin>
```

2) 编写Spark Driver代码

```
package com.baizhi.test1

import org.apache.spark.{SparkConf, SparkContext}

object SparkWordCount {

  def main(args: Array[String]): Unit = {

    //1.创建SparkContext
    val sparkConf: SparkConf =
          new SparkConf().setMaster("spark://spark1:7077").setAppName("wc")
    val sc = new SparkContext(sparkConf)

    //2.读取文件内容，执行SparkRDD任务
    sc.textFile("hdfs://spark1:9000/a.txt")
      .flatMap(_.split(" "))
      .map((_, 1))
      .groupBy(_._1)
      .map(t => (t._1, t._2.size))
      .saveAsTextFile("hdfs://spark1:9000/results")

    //3.关闭SparkContext
```

```
    sc.stop()
  }
}
```

4) 执行`mvn pacake`指令打包程序

5) 将打包的程序上传到远程集群执行以下脚本

```
[root@spark1 spark]# bin/spark-submit --master spark://spark1:7077 --class
com.baizhi.test1.SparkWordCount /opt/spark-day1-1.0-SNAPSHOT.jar
```

> spark-submit方式适用于生产环境，当然spark也支持本地测试，无需构建spark环境即可测试spark代码

# 四、本地模式

> 本地Spark程序调试需要使用local提交模式，即将本机当做运行环境，Master和Worker都为本机。

创建SparkConf的时候设置额外属性，表明本地执行：

```
val conf = new SparkConf().setAppName("wc").setMaster("local[*]")
```

> local: 只启动一个executor
>
> local[k] : 启动k个executor
>
> local[*] : 启动跟cpu数目相同的executor

代码如下

```
package com.baizhi.test1

import org.apache.spark.{SparkConf, SparkContext}

object SparkWordCount {

  def main(args: Array[String]): Unit = {

    //1.创建SparkContext
    val sparkConf: SparkConf =
          new SparkConf().setMaster("local[*]").setAppName("wc")
    val sc = new SparkContext(sparkConf)

    //2.读取文件内容，执行SparkRDD任务
    sc.textFile("file:///f:/datas/a.txt")
      .flatMap(_.split(" "))
      .map((_, 1))
      .groupBy(_._1)
      .map(t => (t._1, t._2.size))
      .saveAsTextFile("file:///f:/datas/results")
```

```
    //3.关闭SparkContext
    sc.stop()
  }
}
```