> 先知道是什么，再去了解为什么

MapReduce入门概述

MapReduce定义

> MapReduce是一个基于Hadoop的分布式运算程序的编程框架

它的核心功能是将用户编写的业务逻辑代码和自带的组件组合成为一个完整的分布式运算程序，并发的运行在Hadoop集群上。

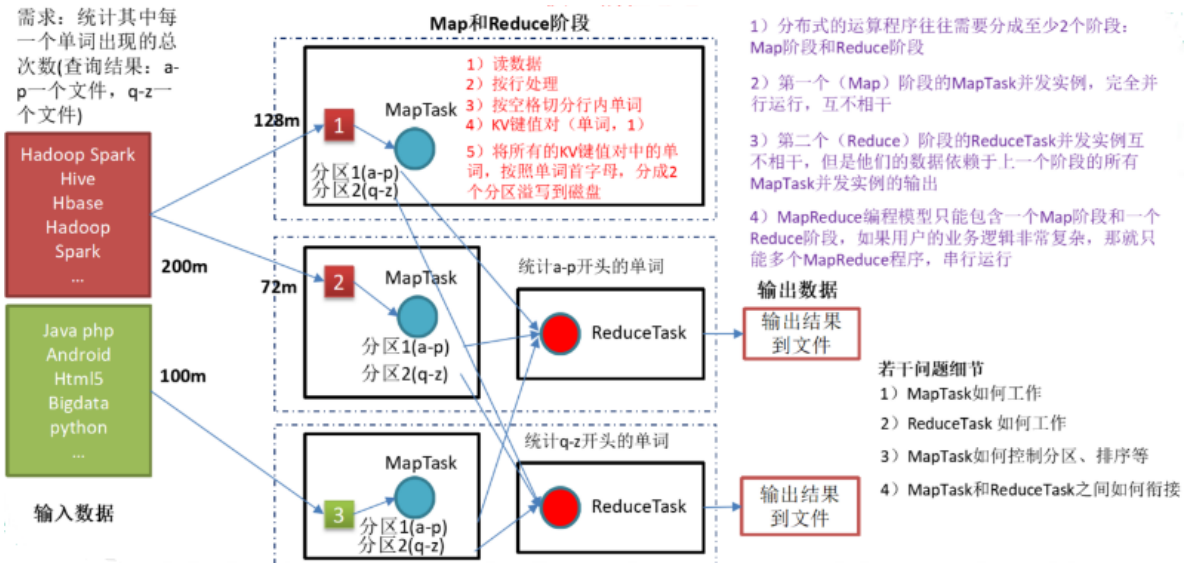
MapReduce优点

- **MapReduce易于编程**：简单的实现一些接口就可以实现分布式程序，并且这个分布式程序可以分布到大量廉价的PC机器上执行。
- **良好的扩展性**：加机器就可以增加计算能力
- **高容错性**：所谓容错就是当系统中一台机器故障时候，有一种机制可以将任务分配到新机器上然后继续运行，这个过程是不需要人工干涉的
- **适合PB级上数据的离线处理**：大数据的稳定处理

MapReduce缺点

- **不擅长实时计算**：MapReduce不能像Mysql，在毫秒级或秒级返回结果
- **不擅长流式计算**：流式计算输入数据是动态的，连续不断的，但是MR处理的数据一定是静态的，这是由设计决定的
- **不擅长DAG计算**：多个任务具有依赖关系，后者输入依赖前者输出，这种活MR不擅长，读写磁盘太多性能下降

MapReduce统计单词过程



默认是按照128M进行数据切块哦

在上图进程一共有三种：

- APPMaster：负责整个程序的过程调度和状态协调
- MapTask：负责Map阶段的整个数据处理流程
- ReduceTask：负责Reduce阶段的整个数据处理流程

【有问题都可以私聊我WX：focusbigdata，或者关注我的公众号：FocusBigData，注意大小写】

MapReduce编程套路

我们编写的部分基本分为三个：Mapper，Reducer和Driver

Map阶段

（1）用户自定义Mapper要继承的父类

（2）Mapper的输入数据格式是KV对

（3）Mapper中业务逻辑写在map()方法中【map()对每个KV对调用一次】

（4）Mapper的输出数据格式也是KV对

Reducer阶段

（1）用户自定义Reducer继承自己的父类

（2）Reducer的输入数据类型对应Mapper的输出数据类型，也是KV

（3）Reduce业务逻辑写在reduce()方法中【reduce()对每个KV对调用一次】

Driver阶段

> 相当于YARN集群的客户端，等等程序写完，需要通过它把整个程序提交到YARN集群上

HELLO WORLD案例

需求

> 统计文件中单词的出现的词频

Mapper代码

```
public class WordcountMapper extends Mapper<LongWritable, Text, Text,
IntWritable>{

    Text k = new Text();
    IntWritable v = new IntWritable(1);

    @Override
    protected void map(LongWritable key, Text value, Context context)   throws
IOException, InterruptedException {

        // 1 获取一行
        String line = value.toString();

        // 2 切割
        String[] words = line.split(" ");
```

```
        // 3 输出
        for (String word : words) {

            k.set(word);
            context.write(k, v);
        }
    }
}
```

Reducer阶段

```
public class WordcountReducer extends Reducer<Text, IntWritable, Text,
IntWritable>{

int sum;
IntWritable v = new IntWritable();

    @Override
    protected void reduce(Text key, Iterable<IntWritable> values,
            Context context) throws IOException, InterruptedException {

        // 1 累加求和
        sum = 0;
        for (IntWritable count : values) {
            sum += count.get();
        }

        // 2 输出
       v.set(sum);
        context.write(key,v);
    }
}
```

Driver驱动类

```
public class WordcountDriver {

    public static void main(String[] args) throws IOException,
ClassNotFoundException, InterruptedException {

// 输入输出路径需要根据自己电脑上实际的输入输出路径设置
// 注意这里是在win下跑，如果放到集群上路径需要更改
args = new String[] { "e:/input/inputword", "e:/output1" };

        // 1 获取配置信息以及封装任务
        Configuration configuration = new Configuration();
        Job job = Job.getInstance(configuration);

        // 2 设置jar加载路径
        job.setJarByClass(WordcountDriver.class);

        // 3 设置map和reduce类
        job.setMapperClass(WordcountMapper.class);
        job.setReducerClass(WordcountReducer.class);

        // 4 设置map输出
```

```
        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(IntWritable.class);

        // 5 设置Reduce输出
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        // 6 设置输入和输出路径
        FileInputFormat.setInputPaths(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        // 7 提交
        boolean result = job.waitForCompletion(true);

        System.exit(result ? 0 : 1);
    }
}
```