# Spark的前世今生

Spark，是一种通用的大数据计算框架，正如传统大数据技术  
Hadoop的MapReduce、 Hive引擎，以及Storm流式实时计算引擎等。  
Spark包含了大数据领域常见的各种计算框架：比如Spark Core用  
于离线计算， Spark SQL用于交互式查询， Spark Streaming用于实时  
流式计算， Spark MLlib用于机器学习， Spark GraphX用于图计算。  
Spark主要用于大数据的计算，而Hadoop以后主要用于大数据的存  
储（比如HDFS、 Hive、 HBase等），以及资源调度（Yarn）。  
Spark+Hadoop的组合，是未来大数据领域最热门的组合，也是最  
有前景的组合！  
**Spark是什么？**  
Spark，是一种"One Stack to rule them all"的大数据计算框架，期望使用一  
个技术堆栈就完美地解决大数据领域的各种计算任务。 Apache官方，对Spark的  
定义就是：通用的大数据快速处理引擎。  
Spark使用Spark RDD、 Spark SQL、 Spark Streaming、 MLlib、 GraphX成  
功解决了大数据领域中，离线批处理、交互式查询、实时流计算、机器学习与图  
计算等最重要的任务和问题。  
Spark除了一站式的特点之外，另外一个最重要的特点，就是基于内存进行  
计算，从而让它的速度可以达到MapReduce、 Hive的数倍甚至数十倍！  
现在已经有很多大公司正在生产环境下深度地使用Spark作为大数据的计算  
框架，包括eBay、 Yahoo!、 BAT、网易、京东、华为、大众点评、优酷土豆、  
搜狗等等。  
Spark同时也获得了多个世界顶级IT厂商的支持，包括IBM、 Intel等。  
**Spark的介绍**  
**Spark整体架构**Spark Engine  
Spark SQL Spark  
Streaming  
GraphX MLlib  
Spark RDD  
Yarn, Mesos, AWS HDFS, S3, Cassandra  
·2009年， Spark诞生于伯克利大学的AMPLab实验室。最出Spark只是  
一个实验性的项目，代码量非常少，属于轻量级的框架。  
·2010年，伯克利大学正式开源了Spark项目。  
·2013年， Spark成为了Apache基金会下的项目，进入高速发展期。第  
三方开发者贡献了大量的代码，活跃度非常高。  
·2014年， Spark以飞快的速度称为了Apache的顶级项目。  
·2015年~， Spark在国内IT行业变得愈发火爆，大量的公司开始重点部  
署或者使用Spark来替代MapReduce、 Hive、 Storm等传统的大数据计  
算框架。  
**Spark的历史沿革**  
**·**速度快： Spark基于内存进行计算（当然也有部分计算基于磁盘，比如shuffle）。  
**·**容易上手开发： Spark的基于RDD的计算模型，比Hadoop的基于Map-Reduce的计算模型  
要更加易于理解，更加易于上手开发，实现各种复杂功能，比如二次排序、 topn等复杂操作  
时，更加便捷。  
**·**超强的通用性： Spark提供了Spark RDD、 Spark SQL、 Spark Streaming、 Spark MLlib、  
Spark GraphX等技术组件，可以一站式地完成大数据领域的离线批处理、交互式查询、流  
式计算、机器学习、图计算等常见的任务。  
**·**集成**Hadoop**： Spark并不是要成为一个大数据领域的“独裁者”，一个人霸占大数据领域  
所有的“地盘”，而是与Hadoop进行了高度的集成，两者可以完美的配合使用。 Hadoop的  
HDFS、 Hive、 HBase负责存储， YARN负责资源调度； Spark复杂大数据计算。实际上，  
Hadoop+Spark的组合，是一种“double win”的组合。  
**·**极高的活跃度： Spark目前是Apache基金会的顶级项目，全世界有大量的优秀工程师是  
Spark的committer。并且世界上很多顶级的IT公司都在大规模地使用Spark。  
**Spark的特点**  
MapReduce能够完成的各种离线批处理功能，以及常见算法（比如二次排序、 topn等），基于  
Spark RDD的核心编程，都可以实现，并且可以更好地、更容易地实现。而且基于Spark RDD编写的离线  
批处理程序，运行速度是MapReduce的数倍，速度上有非常明显的优势。  
Spark相较于MapReduce速度快的最主要原因就在于， MapReduce的计算模型太死板，必须是mapreduce模式，有时候即使完成一些诸如过滤之类的操作，也必须经过map-reduce过程，这样就必须经过  
shuffle过程。而MapReduce的shuffle过程是最消耗性能的，因为shuffle中间的过程必须基于磁盘来读写。  
而Spark的shuffle虽然也要基于磁盘，但是其大量transformation操作，比如单纯的map或者filter等操作，可  
以直接基于内存进行pipeline操作，速度性能自然大大提升。  
但是Spark也有其劣势。由于Spark基于内存进行计算，虽然开发容易，但是真正面对大数据的时候  
（比如一次操作针对10亿以上级别），在没有进行调优的情况下，可能会出现各种各样的问题，比如OOM  
内存溢出等等。导致Spark程序可能都无法完全运行起来，就报错挂掉了，而MapReduce即使是运行缓慢，  
但是至少可以慢慢运行完。  
此外， Spark由于是新崛起的技术新秀，因此在大数据领域的完善程度，肯定不如MapReduce，比如  
基于HBase、 Hive作为离线批处理程序的输入输出， Spark就远没有MapReduce来的完善。实现起来非常  
麻烦。  
**Spark VS MapReduce**  
Spark SQL实际上并不能完全替代Hive，因为Hive是一种基于HDFS的数据仓库，并且提供了基于  
SQL模型的，针对存储了大数据的数据仓库，进行分布式交互查询的查询引擎。  
严格的来说， Spark SQL能够替代的，是Hive的查询引擎，而不是Hive本身，实际上即使在生产环境  
下， Spark SQL也是针对Hive数据仓库中的数据进行查询， Spark本身自己是不提供存储的，自然也不可能  
替代Hive作为数据仓库的这个功能。  
Spark SQL的一个优点，相较于Hive查询引擎来说，就是速度快，同样的SQL语句，可能使用Hive的  
查询引擎，由于其底层基于MapReduce，必须经过shuffle过程走磁盘，因此速度是非常缓慢的。很多复杂  
的SQL语句，在hive中执行都需要一个小时以上的时间。而Spark SQL由于其底层基于Spark自身的基于内  
存的特点，因此速度达到了Hive查询引擎的数倍以上。  
但是Spark SQL由于与Spark一样，是大数据领域的新起的新秀，因此还不够完善，有少量的Hive支持  
的高级特性， Spark SQL还不支持，导致Spark SQL暂时还不能完全替代Hive的查询引擎。而只能在部分  
Spark SQL功能特性可以满足需求的场景下，进行使用。  
而Spark SQL相较于Hive的另外一个优点，就是支持大量不同的数据源，包括hive、  
json、 parquet、 jdbc等等。此外， Spark SQL由于身处Spark技术堆栈内，也是基于RDD来  
工作，因此可以与Spark的其他组件无缝整合使用，配合起来实现许多复杂的功能。比如  
Spark SQL支持可以直接针对hdfs文件执行sql语句！  
**Spark SQL VS Hive**  
Spark Streaming与Storm都可以用于进行实时流计算。但是他们两者的区别是非常大的。其中区别之一，就是， Spark Streaming和Storm的计算模型完全不一样， Spark Streaming是基于RDD的，因此需要将一小段时间内的，比如1秒内的数据，收集起来，作为一个RDD，然后再针对这个batch的数据进行处理。  
而Storm却可以做到每来一条数据，都可以立即进行处理和计算。因此， Spark Streaming实际上严格意义上来说，只能称作准实时的流计算框架；而Storm是真正意义上的实时计算框架。  
此外， Storm支持的一项高级特性，是Spark Streaming暂时不具备的，即Storm支持在分布式流式计算程序（Topology）在运行过程中，可以动态地调整并行度，从而动态提高并发处理能力。而SparkStreaming是无法动态调整并行度的。  
但是Spark Streaming也有其优点，首先Spark Streaming由于是基于batch进行处理的，因此相较于Storm基于单条数据进行处理，具有数倍甚至数十倍的吞吐量。  
此外， Spark Streaming由于也身处于Spark生态圈内，因此Spark Streaming可以与Spark Core、Spark SQL，甚至是Spark MLlib、 Spark GraphX进行无缝整合。流式处理完的数据，可以立即进行各种map、 reduce转换操作，可以立即使用sql进行查询，甚至可以立即使用machine learning或者图计算算法  
进行处理。这种一站式的大数据处理功能和优势，是Storm无法匹敌的。  
因此，综合上述来看，通常在对实时性要求特别高，而且实时数据量不稳定，比如在白天有高峰期的情况下，可以选择使用Storm。但是如果是对实时性要求一般，允许1秒的准实时处理，而且不要求动态调整并行度的话，选择Spark Streaming是更好的选择。  
**Spark Streaming VS Storm**  
首先， Spark目前来说，相较于MapReduce来说，可以立即替代的，并且会产生非常理想的效果的场景，就是要求低延时的复杂大数据交互式计算系统。比如某些大数据系统，可以根据用户提交的各种条件，立即定制执行复杂的大数据计算系统，并且要求低延时（一小时以内）即可以出来结果，并通过前端页面  
展示效果。在这种场景下，对速度比较敏感的情况下，非常适合立即使用Spark替代MapReduce。因为Spark编写的离线批处理程序，如果进行了合适的性能调优之后，速度可能是MapReduce程序的十几倍。

从而达到用户期望的效果。  
其次，相对于Hive来说，对于某些需要根据用户选择的条件，动态拼接SQL语句，进行某类特定查询统计任务的系统，其实类似于上述的系统。此时也要求低延时，甚至希望达到几分钟之内。此时也可以使用Spark SQL替代Hive查询引擎。因此场景比较固定， SQL语句的语法比较固定，清楚肯定不会使用到  
Spark SQL所不支持的Hive语法特性。此时使用Hive查询引擎可以需要几十分钟执行一个复杂SQL。而使用Spark SQl，可能只需要使用几分钟。可以达到用户期望的效果最后，对于Storm来说，如果仅仅要求对数据进行简单的流式计算处理，那么选择storm或者sparkstreaming都无可厚非。但是如果需要对流式计算的中间结果（RDD），进行复杂的后续处理，则使用Spark更好，因为Spark本身提供了很多原语，比如map、 reduce、 groupByKey、 filter等等。  
**Spark的个人使用体会**  
Spark目前在国内正在飞速地发展，并且在很多领域，以及慢慢开始替代传统得一些基于Hadoop的组件。比如BAT、京东、搜狗等知名的互联网企业，都在深度的，大规模地使用Spark。  
但是，大家如果去观察一下一些招聘网站对大数据的招聘需求，就会发现，目前来说，由于大部分还是大公司在使用Spark，因此大部分中小型企业，还是主要在使用Hadoop进行大数据处理。在招聘时，还是主要以hadoop工程师为主。 Spark以及Storm的招聘还是相对Hadoop来说，会少一些。  
但是，大家如果通过本堂课的讲解，能够较为全面地对Spark有一个感性得认识，就能意识到， Spark在大数据领域中，是未来的一个趋势和方向。随着Spark、 Spark SQL以及Spark Streaming慢慢成熟，就会慢慢替代掉Hadoop的MapReduce、 Hive查询等。大家可以想想，如果两者都能够实现相同的功能，而Spark甚至以后还可以做的更好，速度要快好几倍，甚至好几十倍。那么还有谁会愿意使用MapReduce或Hive查询引擎呢？  
实际上，根据我在国内一线互联网公司这几年的工作和观察，以及通过与行业内各个规模公司的朋友交流，认为，未来的主流，一定是hadoop+Spark的这种组合， double win的格局。 hadoop的特长，就是hdfs，分布式存储，基于此之上的是Hive作为大数据的数据仓库， HBase作为大数据的实时查询NoSQL数据库， YARN作为通用的资源调度框架；而Spark，则发挥它的特长，将各种各样的大数据计算模型汇聚在一个技术堆栈内，对hadoop上的大数据进行各种计算处理！  
因此，大家也可以看到， Spark目前正在变得越来越火爆，招聘的企业正在越来越多，而且目前国内spark人才可以说是稀缺！！！在目前，以及未来，完全供不应求！因此这种趋势，以及这种现状，就决定了，对于我们个人来说，目前进行spark的学习以及研究，完全是未来一个获取快速升值的机会！！！  
**Spark目前在国内的现状以及未来的展望**  
  
  
  
课程介绍、特色与价值  
  
  
  
·Spark的前世今生。  
·课程介绍、特色与价值。  
·Scala编程详解。 为什么要学习Scala？阅读Spark源码、在公司需要的时候使用  
Scala进行Spark应用的开发，但是本课程所有示例代码用Java  
·Scala基础语法  
·Scala面向对象编程  
·Scala函数式编程  
·Scala高级特性（泛型、隐式转换）  
·Scala Actor并发编程  
·课程环境搭建。  
·Hadoop 2.4.1集群搭建  
·Spark 1.3.0集群搭建  
**课程内容介绍：课程前置部分**  
·RDD介绍  
·Spark基本工作原理  
·Spark开发入门  
·编写WordCount程序  
·使用本地模式进行测试  
·使用spark-submit提交到集群运行（spark-submit常用参数说明）  
·Spark程序开发流程总结  
·spark-shell的使用（编写wordcount程序）  
·创建RDD：并行化集合、基于文件创建RDD  
·操作RDD： transformation和action， java 8和旧版本的区别，操作key-value对  
·RDD常用操作全程案例实战  
·RDD持久化： cache()和persist()，几种持久化策略  
·共享变量： broadcast variable、 accumulator  
·RDD高级编程：基于排序算法的WordCount、二次排序、 topn、 combineByKey  
**课程内容介绍：Spark核心编程**  
·Spark内核概览  
·Spark核心概念  
·Spark工作流程  
·Spark运行模式  
·SparkContext原理剖析与源码分析  
·job触发流程原理剖析与源码分析  
·Master原理剖析（资源调度算法）  
·高可用机制原理剖析  
·注册机制原理剖析  
·executor失败容错机制原理剖析  
·资源调度算法剖析  
·Worker原理剖析  
**课程内容介绍：结合源码深度剖析Spark内核**  
·DAGScheduler原理剖析  
·stage划分算法  
·TaskScheduler原理剖析  
·task分配算法  
·Executor原理剖析  
·ShuffleMapTask和ResultTask原理剖析  
·Shuffle原理剖析  
·Storage模块原理剖析  
·BlockManager原理剖析  
·Cache原理剖析  
·Checkpoint原理剖析  
**课程内容介绍：结合源码深度剖析Spark内核**  
·使用Kryo进行序列化  
·优化数据结构  
·对多次执行action operation的RDD进行持久化  
·对RDD持久化进行序列化  
·垃圾回收调优  
·提高并行度  
·广播大数据集  
·数据本地化  
·reduceByKey和groupByKey  
·shuffle性能调优  
**课程内容介绍：Spark性能优化**  
·DataFrame的使用  
·将RDD转化为DataFrame  
·支持的数据源（parquet、 json、 hive、 jdbc）  
·工作原理  
·性能调优  
**课程内容介绍：Spark SQL**  
·基本工作原理  
·WordCount与开发流程  
·输入DStream（hdfs、 socket、 kafka）  
·DStream的transformation操作（updateStateByKey、 transform、 slide window）  
·DStream的output操作（性能优化与最佳实践）  
·Spark Streaming与Spark SQL整合  
·Cache、 Checkpoint、 Ahead Write Log  
·容错机制  
·源码剖析  
·性能调优  
**课程内容介绍：Spark Streaming**  
MLLib和Graphx本系列课程不讲。 MLlib和Graphx分别用于机器学习和图计算。  
一是因为，在目前国内，大部分的Spark开发岗位中，其实主要还是使用Spark Core、  
Spark SQL和Spark Streaming，很少使用MLlib和Graphx。因此就算讲了，也未必就一定马  
上会有价值。在市场上对MLlib和Graphx的需求量，是非常少的，通常都是专业的机器学习  
工程师会使用。  
二是因为，机器学习和图计算本身都涵盖非常多的，和深奥的专业知识，本系列课程的目标  
是让Spark开发人员能够从入门到精通，总共就几十讲的时间，如果还讲这两个东西，会耗  
费大量时间。最后就导致Spark的组件中没有一个是讲透彻的，都是泛泛入门。  
因此，本系列课程的定位就是，让Spark开发人员能够零基础起步，从入门到精通Spark  
Core、 Spark SQL和Spark Streaming的开发。而不会涉及MLlib和Graphx。用50~60讲的时  
间把核心开发相关的三个组件彻底从源码的角度讲透彻！  
MLlib和Graphx，如果未来有时间，有机会，再单独用系列课程讲解。  
**课程内容介绍：MLlib和GraphX**  
**课程内容介绍：各个部分的内容学习好的效果**·如果能够学扎实基础课程，以及Spark核心编程，那么可以称之为Spark入门级别的水平。  
·如果能够学扎实基础课程、 Spark核心编程，以及Spark SQL和Spark  
Streaming的所有功能使用，并熟练掌握，那么可以称之为熟悉Spark的水平。  
·如果能够学精通本课程所有的内容，包括基础、各组件功能使用、 Spark内核原理、 Spark内核源码、 Spark性能调优、 Spark SQL原理和性能调优、 Spark  
Streaming原理和性能调优，那么可以称之为精通Spark的水平。  
根据我在企业中面试Spark工程师的经验来看，应届生，需要达到入门级的水平，去面试校招； 1~3年工作经验的，需要达到熟练的水平去面试Spark开发工程师的岗位； 3年以上工作经验的，需要达到精通Spark的水平，去面试Spark高级开发工程师的岗位。  
·使用最新版本： Hadoop 2.4.1、 Spark 1.3.0（其他课程都是Spark 1.1及以前的版本）  
·从零起步： 从scala到环境手把手搭建到精通spark开发和源码（其他课程很多都省略了环境  
搭建等步骤，导致零基础者无从下手）  
·涵盖Spark所有功能（其他课程一般都只包含了Spark的部分基础功能，不涉及高级功能，  
比如spark streaming容错、二次排序、 combineByKey等）  
·全程案例实战： 所有功能均基于案例实战驱动（其他课程很多都是写几个简单demo）  
·结合源码对Spark内核进行深度剖析：彻底讲透Spark内核（其他课程虽然也讲内核，但都  
是浅尝辄止，讲的浅，讲不透，让人云里雾里，更不用说结合源码了）  
·对原理以及内核部分全程手工画图讲解（其他课程都是对着PPT，或者网络上已有的图片，  
干讲内核，让人不好理解）  
·深度讲解Spark性能调优，尤其是精通spark shuffle调优（其它课程基本都是讲一些最基础的优化方法）  
**课程特色**  
·Java / J2EE开发工程师  
·Hadoop开发工程师  
·Spark入门级别的，或者只有一定基础的  
·在校或者刚毕业的学生  
·对于Java/J2EE开发工程师，可以通过学习本课程进行转型，成功转型为大数据领域的工程师，当然，前提是，建议自己补充Hadoop基础的知识。普通J2EE工程师的薪资其实一般都在20k以下，但是如果有2~3年工作经验的人，达到熟悉或者精通本课程的水平，达到20k~30k是绝对没有问题的。  
·对于Hadoop开发工程师，以及Spark入门级的，可以通过学习本课程，增加自己在大数据  
领域的技能，提高自己在公司，在职场的竞争力。争取在公司内的升级、升职、加薪，承接公司最新的基于Spark的项目。当然，也完全可以通过学习，进行跳槽。  
·对于在校或者刚毕业的学生，如果有志进入大数据行业，则可以通过学习本课程，增加自己在校招中的通过率，提高自己简历的含金量，为自己获取更多的面试机会。并且提高自己刚毕业的薪资。  
**课程面向的人群以及课程的价值**

# 第28讲-Spark核心编程： Spark基本工作原理与RDD

画图讲解Spark的基本工作原理  
1、分布式  
2、主要基于内存（少数情况基于磁盘）  
3、迭代式计算  
**Spark基本工作原理**  
1、 RDD是Spark提供的核心抽象，全称为Resillient Distributed Dataset，即弹性分布式数据集。  
2、 RDD在抽象上来说是一种元素集合，包含了数据。它是被分区的，分为多个  
分区，每个分区分布在集群中的不同节点上，从而让RDD中的数据可以被并行  
操作。（分布式数据集）  
3、 RDD通常通过Hadoop上的文件，即HDFS文件或者Hive表，来进行创建；有时也可以通过应用程序中的集合来创建。  
4、 RDD最重要的特性就是，提供了容错性，可以自动从节点失败中恢复过来。  
即如果某个节点上的RDD partition，因为节点故障，导致数据丢了，那么RDD  
会自动通过自己的数据来源重新计算该partition。这一切对使用者是透明的。  
5、 RDD的数据默认情况下存放在内存中的，但是在内存资源不足时， Spark会自动将RDD数据写入磁盘。（弹性）  
**RDD以及其特点**  
1、核心开发：离线批处理 / 延迟性的交互式数据处理  
2、 SQL查询：底层都是RDD和计算操作  
3、实时计算：底层都是RDD和计算操作  
**什么是Spark开发？**

# 第29讲-Spark核心编程：使用Java、 Scala和spark-shell开发wordcount程序

1、用Java开发wordcount程序  
1.1 配置maven环境  
1.2 如何进行本地测试  
1.3 如何使用spark-submit提交到spark集群进行执行（spark-submit常用参数说明， sparksubmit其实就类似于hadoop的hadoop jar命令）  
2、用Scala开发wordcount程序  
2.1 下载scala ide for eclipse  
2.2 在Java Build Path中，添加spark依赖包（如果与scala ide for eclipse原生的scala版本  
发生冲突，则移除原生的scala / 重新配置scala compiler）  
2.3 用export导出scala spark工程  
3、用spark-shell开发wordcount程序  
3.1 常用于简单的测试  
**开发wordcount程序**

# 第30讲-Spark核心编程： wordcount程序原理深度剖析

val conf = new SparkConf().setAppName("WordCount")  
val sc = new JavaSparkContext(conf)  
val lines = sc.textFile("hdfs://spark1:9000/spark.txt")  
val words = lines.flatMap(line => line.split(" "))  
val pairs = words.map(word => (word, 1))  
val wordCounts = pairs.reduceByKey(\_ + \_)  
wordCounts.foreach(wordCount => println(wordCount.\_1 + " appears " +  
wordCount.\_2 + " times."))  
**wordcount程序**

# 第31讲-Spark核心编程： Spark架构原理

1、 Driver  
2、 Master  
3、 Worker  
4、 Executor  
5、 Task  
**Spark架构原理**

# 第32讲-Spark核心编程：创建RDD（集合、本地文件、 HDFS文件）

进行Spark核心编程时，首先要做的第一件事，就是创建一个初始的RDD。该RDD中，通常就代表和包含了Spark应用程序的输入源数据。然后在创建了初始的RDD之后，才可以通过Spark Core提供的transformation算子，对该RDD进行转换，来获取其他的RDD。  
Spark Core提供了三种创建RDD的方式，包括：使用程序中的集合创建RDD；使用本地文件创建RDD；使用HDFS文件创建RDD。  
个人经验认为：  
1、使用程序中的集合创建RDD，主要用于进行测试，可以在实际部署到集群运行之前，自己使用集合构造测试数据，来测试后面的spark应用的流程。  
2、使用本地文件创建RDD，主要用于临时性地处理一些存储了大量数据的文件。  
3、使用HDFS文件创建RDD，应该是最常用的生产环境处理方式，主要可以针对HDFS上存储的大数据，进行离线批处理操作。  
**创建RDD**  
如果要通过并行化集合来创建RDD，需要针对程序中的集合，调用SparkContext的parallelize()方法。 Spark会将集合中的数据拷贝到集群上去，形成一个分布式的数据集合，也就是一个RDD。相当于是，集合中的部分数据会到一个节点上，而另一部分数据会到其他节点上。然后就可以用并行的方式来操作这个分布式数据集合，即RDD。  
// 案例： 1到10累加求和  
val arr = Array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)  
val rdd = sc.parallelize(arr)  
val sum = rdd.reduce(\_ + \_)  
调用parallelize()时，有一个重要的参数可以指定，就是要将集合切分成多少个partition。  
Spark会为每一个partition运行一个task来进行处理。 Spark官方的建议是，为集群中的每个CPU创建2~4个partition。 Spark默认会根据集群的情况来设置partition的数量。但是也可以在调用parallelize()方法时，传入第二个参数，来设置RDD的partition数量。比如parallelize(arr, 10)  
**并行化集合创建RDD**  
Spark是支持使用任何Hadoop支持的存储系统上的文件创建RDD的，比如说HDFS、Cassandra、 HBase以及本地文件。通过调用SparkContext的textFile()方法，可以针对本地文件或HDFS文件创建RDD。  
有几个事项是需要注意的：  
1、如果是针对本地文件的话，如果是在windows上本地测试， windows上有一份文件即可；  
如果是在spark集群上针对linux本地文件，那么需要将文件拷贝到所有worker节点上。  
2、 Spark的textFile()方法支持针对目录、压缩文件以及通配符进行RDD创建。  
3、 Spark默认会为hdfs文件的每一个block创建一个partition，但是也可以通过textFile()的第二个参数手动设置分区数量，只能比block数量多，不能比block数量少。  
// 案例：文件字数统计  
val rdd = sc.textFile("data.txt")  
val wordCount = rdd.map(line => line.length).reduce(\_ + \_)  
**使用本地文件和HDFS创建RDD**  
Spark的textFile()除了可以针对上述几种普通的文件创建RDD之外，还有一些特列的方法来创建RDD：  
1、 SparkContext.wholeTextFiles()方法，可以针对一个目录中的大量小文件，返回<filename, fileContent>组成的pair，作为一个PairRDD，而不是普通的RDD。普通的textFile()返回的RDD中，每个元素就是文件中的一行文本。  
2、 SparkContext.sequenceFile[K, V]()方法，可以针对SequenceFile创建RDD， K和V泛型类型就是SequenceFile的key和value的类型。 K和V要求必须是Hadoop的序列化类型，比如IntWritable、 Text等。  
3、 SparkContext.hadoopRDD()方法，对于Hadoop的自定义输入类型，可以创建RDD。该方法接收JobConf、 InputFormatClass、 Key和Value的Class。  
4、 SparkContext.objectFile()方法，可以针对之前调用RDD.saveAsObjectFile()创建的对象序列化的文件，反序列化文件中的数据，并创建一个RDD。  
**使用本地文件和HDFS创建RDD**

# 第33讲-Spark核心编程：操作RDD（transformation和action案例实战）

Spark支持两种RDD操作： transformation和action。 transformation操作会针对已有的RDD创建一个新的RDD；而action则主要是对RDD进行最后的操作，比如遍历、 reduce、保存到文件等，并可以返回结果给Driver程序。  
例如， map就是一种transformation操作，它用于将已有RDD的每个元素传入一个自定义的函数，并获取一个新的元素，然后将所有的新元素组成一个新的RDD。而reduce就是一种action操作，它用于对RDD中的所有元素进行聚合操作，并获取一个最终的结果，然后返回给Driver程序。  
transformation的特点就是lazy特性。 lazy特性指的是，如果一个spark应用中只定义了transformation操作，那么即使你执行该应用，这些操作也不会执行。也就是说， transformation是不会触发spark程序的执行的，它们只是记录了对RDD所做的操作，但是不会自发的执行。只有当transformation之后，接着执行了一个action操作，那么所有的transformation才会执行。 Spark通过这种lazy特性，来进行底层的spark应用执行的优化，避免产生过多中间结果。  
action操作执行，会触发一个spark job的运行，从而触发这个action之前所有的transformation的执行。这是action的特性。  
**transformation和action介绍**  
这里通过一个之前学习过的案例，统计文件字数，来讲解transformation和action。  
// 这里通过textFile()方法，针对外部文件创建了一个RDD， lines，但是实际上，程序执行到这里为止，spark.txt文件的数据是不会加载到内存中的。 lines，只是代表了一个指向spark.txt文件的引用。  
val lines = sc.textFile("spark.txt")  
// 这里对lines RDD进行了map算子，获取了一个转换后的lineLengths RDD。但是这里连数据都没有，当然也不会做任何操作。 lineLengths RDD也只是一个概念上的东西而已。  
val lineLengths = lines.map(line => line.length)  
// 之列，执行了一个action操作， reduce。此时就会触发之前所有transformation操作的执行， Spark会将操作拆分成多个task到多个机器上并行执行，每个task会在本地执行map操作，并且进行本地的reduce聚合。  
最后会进行一个全局的reduce聚合，然后将结果返回给Driver程序。  
val totalLength = lineLengths.reduce(\_ + \_)**案例：统计文件字数**  
Spark有些特殊的算子，也就是特殊的transformation操作。比如groupByKey、 sortByKey、reduceByKey等，其实只是针对特殊的RDD的。即包含key-value对的RDD。而这种RDD中的元素，实际上是scala中的一种类型，即Tuple2，也就是包含两个值的Tuple。  
在scala中，需要手动导入Spark的相关隐式转换， import org.apache.spark.SparkContext.\_。  
然后，对应包含Tuple2的RDD，会自动隐式转换为PairRDDFunction，并提供reduceByKey等方法。  
val lines = sc.textFile("hello.txt")  
val linePairs = lines.map(line => (line, 1))  
val lineCounts = linePairs.reduceByKey(\_ + \_)  
lineCounts.foreach(lineCount => println(lineCount.\_1 + " appears " + llineCount.\_2 + "times."))  
**案例：统计文件每行出现的次数**  
**常用transformation介绍**

|  |  |
| --- | --- |
| 操作 | 介绍 |
| map | 将RDD中的每个元素传入自定义函数，获取 一个新的元素，然后用新的元素组成新的RDD |
| filter | 对RDD中每个元素进行判断，如果返回true 则保留，返回false则剔除。 |
| flatMap | 与map类似，但是对每个元素都可以返回一 个或多个新元素。 |
| gropuByKey | 根据key进行分组，每个key对应一个 Iterable<value> |
| reduceByKey | 对每个key对应的value进行reduce操作。 |
| sortByKey | 对每个key对应的value进行排序操作。 |
| join | 对两个包含<key,value>对的RDD进行join操作 ，每个key join上的pair，都会传入自定义函数 进行处理。 |
| cogroup | 同join，但是是每个key对应的 Iterable<value>都会传入自定义函数进行处理 。 |

**常用action介绍**

|  |  |
| --- | --- |
| 操作 | 介绍 |
| reduce | 将RDD中的所有元素进行聚合操作。第一个 和第二个元素聚合，值与第三个元素聚合， 值与第四个元素聚合，以此类推。 |
| collect | 将RDD中所有元素获取到本地客户端。 |
| count | 获取RDD元素总数。 |
| take(n) | 获取RDD中前n个元素。 |
| saveAsTextFile | 将RDD元素保存到文件中，对每个元素调用 toString方法 |
| countByKey | 对每个key对应的值进行count计数。 |
| foreach | 遍历RDD中的每个元素。 |

# 第34讲-Spark核心编程： transformation操作开发实战

1、 map：将集合中每个元素乘以2  
2、 filter：过滤出集合中的偶数  
3、 flatMap：将行拆分为单词  
4、 groupByKey：将每个班级的成绩进行分组  
5、 reduceByKey：统计每个班级的总分  
6、 sortByKey：将学生分数进行排序  
7、 join：打印每个学生的成绩  
8、 cogroup：打印每个学生的成绩  
**transformation操作开发实战**

# 第35讲-Spark核心编程： action操作开发实战

1、 reduce：  
2、 collect：  
3、 count：  
4、 take：  
5、 saveAsTextFile：  
6、 countByKey：  
7、 foreach：

**action操作开发实战**

# 第36讲-Spark核心编程： RDD持久化详解

Spark非常重要的一个功能特性就是可以将RDD持久化在内存中。当对RDD执行持久化操作时，每个节点都会将自己操作的RDD的partition持久化到内存中，并且在之后对该RDD的反复使用中，直接使用内存缓  
存的partition。这样的话，对于针对一个RDD反复执行多个操作的场景，就只要对RDD计算一次即可，后面直接使用该RDD，而不需要反复计算多次该RDD。  
巧妙使用RDD持久化，甚至在某些场景下，可以将spark应用程序的性能提升10倍。对于迭代式算法和快速交互式应用来说， RDD持久化，是非常重要的。  
要持久化一个RDD，只要调用其cache()或者persist()方法即可。在该RDD第一次被计算出来时，就会直接缓存在每个节点中。而且Spark的持久化机制还是自动容错的，如果持久化的RDD的任何partition丢失了，那么Spark会自动通过其源RDD，使用transformation操作重新计算该partition。  
cache()和persist()的区别在于， cache()是persist()的一种简化方式， cache()的底层就是调用的persist()的无参版本，同时就是调用persist(MEMORY\_ONLY)，将数据持久化到内存中。如果需要从内存中清楚缓存，那么可以使用unpersist()方法。  
Spark自己也会在shuffle操作时进行数据的持久化，比如写入磁盘，主要是为了在节点失败时，避免需要重新计算整个过程。  
**RDD持久化原理**  
实际编码体验RDD持久化的使用，以及其效果。  
**RDD持久化实战**  
RDD持久化是可以手动选择不同的策略的。比如可以将RDD持久化在内存中、持久化到磁盘上、使用序列化的方式持久化，多持久化的数据进行多路复用。只要在调用persist()时传入对应的StorageLevel即可。  
**RDD持久化策略**

|  |  |
| --- | --- |
| 持久化级别 | 含义 |
| MEMORY\_ONLY | 以非序列化的Java对象的方式持久化在JVM 内存中。如果内存无法完全存储RDD所有的 partition，那么那些没有持久化的partition就 会在下一次需要使用它的时候，重新被计算 。 |
| MEMORY\_AND\_DISK | 同上，但是当某些partition无法存储在内存 中时，会持久化到磁盘中。下次需要使用这 些partition时，需要从磁盘上读取。 |
| MEMORY\_ONLY\_SER | 同MEMORY\_ONLY，但是会使用Java序列化 方式，将Java对象序列化后进行持久化。可以 减少内存开销，但是需要进行反序列化，因 此会加大CPU开销。 |

RDD持久化是可以手动选择不同的策略的。比如可以将RDD持久化在内存中、持久化到磁盘上、使用序列化的方式持久化，多持久化的数据进行多路复用。只要在调用persist()时传入对应的StorageLevel即可。  
**RDD持久化策略**

|  |  |
| --- | --- |
| 持久化级别 | 含义 |
| MEMORY\_AND\_DSK\_SER | 同MEMORY\_AND\_DSK。但是使用序列化方 式持久化Java对象。 |
| DISK\_ONLY | 使用非序列化Java对象的方式持久化，完全 存储到磁盘上。 |
| MEMORY\_ONLY\_2 MEMORY\_AND\_DISK\_2 等等 | 如果是尾部加了2的持久化级别，表示会将 持久化数据复用一份，保存到其他节点，从 而在数据丢失时，不需要再次计算，只需要 使用备份数据即可。 |

Spark提供的多种持久化级别，主要是为了在CPU和内存消耗之间进行取舍。下面是一些通用的持久化级别的选择建议：  
1、优先使用MEMORY\_ONLY，如果可以缓存所有数据的话，那么就使用这种策略。因为纯内存速度最快，而且没有序列化，不需要消耗CPU进行反序列化操作。  
2、如果MEMORY\_ONLY策略，无法存储的下所有数据的话，那么使用  
MEMORY\_ONLY\_SER，将数据进行序列化进行存储，纯内存操作还是非常快，只是要消耗CPU进行反序列化。  
3、如果需要进行快速的失败恢复，那么就选择带后缀为\_2的策略，进行数据的备份，这样在失败时，就不需要重新计算了。  
4、能不使用DISK相关的策略，就不用使用，有的时候，从磁盘读取数据，还不如重新计算一次。  
**如何选择RDD持久化策略？**

# 第37讲-Spark核心编程：共享变量（Broadcast Variable和Accumulator）

Spark一个非常重要的特性就是共享变量。  
默认情况下，如果在一个算子的函数中使用到了某个外部的变量，那么这个变量  
的值会被拷贝到每个task中。此时每个task只能操作自己的那份变量副本。如果多个task想要共享某个变量，那么这种方式是做不到的。  
Spark为此提供了两种共享变量，一种是Broadcast Variable（广播变量），另一  
种是Accumulator（累加变量）。 Broadcast Variable会将使用到的变量，仅仅为每个节点拷贝一份，更大的用处是优化性能，减少网络传输以及内存消耗。  
Accumulator则可以让多个task共同操作一份变量，主要可以进行累加操作。  
**共享变量工作原理**  
Spark提供的Broadcast Variable，是只读的。并且在每个节点上只会有一份副本，而不会为每个task都拷贝一份副本。因此其最大作用，就是减少变量到各个节点的网络传输消耗，以及在各个节点上的内存消耗。  
此外， spark自己内部也使用了高效的广播算法来减少网络消耗。  
可以通过调用SparkContext的broadcast()方法，来针对某个变量创建广播变量。然后在算子的函数内，使用到广播变量时，每个节点只会拷贝一份副本了。每个节点可以使用广播变量的value()方法获取值。记住，广播变量，是只读的。  
val factor = 3  
val factorBroadcast = sc.broadcast(factor)  
val arr = Array(1, 2, 3, 4, 5)  
val rdd = sc.parallelize(arr)  
val multipleRdd = rdd.map(num => num \* factorBroadcast.value())  
multipleRdd.foreach(num => println(num))  
**Broadcast Variable**  
Spark提供的Accumulator，主要用于多个节点对一个变量进行共享性的操作。 Accumulator只提供了累加的功能。但是确给我们提供了多个task对一个变量并行操作的功能。但是task只能对Accumulator进行累加操作，不能读取它的值。只有Driver程序可以读取Accumulator的值。  
val sumAccumulator = sc.accumulator(0)  
val arr = Array(1, 2, 3, 4, 5)  
val rdd = sc.parallelize(arr)  
rdd.foreach(num => sumAccumulator += num)  
println(sumAccumulator.value)  
**Accumulator**

# 第38讲-Spark核心编程：高级编程之基于排序机制的wordcount程序

1、对文本文件内的每个单词都统计出其出现的次数。  
2、按照每个单词出现次数的数量，降序排序。  
**案**

# 第39讲-Spark核心编程：高级编程之二次排序

1、按照文件中的第一列排序。  
2、如果第一列相同，则按照第二列排序。  
**案例需求**

# 第40讲-Spark核心编程：高级编程之topn 1、对文本文件内的数字，取最大的前3个。 2、对每个班级内的学生成绩，取出前3名。（分组取topn） 3、课后作用：用Scala来实现分组取topn。 案例需求